NASA-CR-192960

TECHNICAL REPORT

Techniques for Designing Rotorcraft Control Systems

William S. Levine and Jewel Barlow
Principal Investigators
Departments of Electrical and Aeronautical
Engineering and Institute for Systems Research
University of Maryland
College Park, MD 20742

April 1, 1992 - March 31, 1993

NASA Ames NAG 2-794

Rotorcraft Flight Control Design Using CONSOL-OPTCAD

Annual Report

Gil Yudilevitch William S. Levine

April 5, 1993

1 Introduction

This report summarizes the work that has been done on the project from April 1, 1992 to March 31, 1993. The main goal of this research is to develop a practical tool for rotorcraft control system design based on interactive optimization tools (CONSOL-OPTCAD) as well as classical rotorcraft design considerations (ADOCS). This approach enables the designer to combine engineering intuition and experience with parametric optimization. The combination should make it possible to produce better design faster than would be possible using either pure optimization or pure intuition and experience.

We emphasize that the goal of this project is not to develop an algorithm. It is to develop a tool. We want to keep the human designer in the design process so as to be able to take advantage of his or her experience and creativity. The role of the computer is to perform the calculation necessary to improve, and to display the performance of the nominal design.

Briefly, during the first year we have connected CONSOL-OPTCAD, an existing software package for optimizing parameters with respect to multiple performance criteria, to a simplified nonlinear simulation of the UH-60 rotorcraft. We have also created mathematical approximations to the Mil-specs for rotorcraft handling qualities and input them into CONSOL-OPTCAD. Finally, we have developed the additional software necessary to use CONSOL-OPTCAD for the design of rotorcraft controllers.

In order to meet the specification we do not actually have to solve an optimization problem (i.e., we are not looking for a global minimum for the cost functions). In fact, we are just looking for a feasible solution, i.e., a solution which satisfies all the mathematical constraints (handling

qualities requirements). Using the optimization package CONSOL-OPTCAD we actually try to solve this problem by *min/max* optimization techniques [1].

If the problem, has a feasible solution, and the performance criteria are convex and smooth (with respect to the design parameters), a solution can be found iteratively by minimizing the maximum of the normalized (weighted) performances (*min/max*). However, in complicated problems, such as design of rotorcraft flight controls, the system performance criteria are usually not convex and not smooth. Moreover, it is not clear whether there exists a feasible solution or not. For these cases we can still use the *min/max* techniques to obtain the best possible solution (i.e., most of the requirements are satisfied and the rest are "almost" satisfied). In order to solve this problem we first have to transfer the system performance criteria and constraints into smooth functions (see Paragraph 2.2). Then we have to choose two sets of parameters. One is the initial guess for the design parameters. The second is the performances weighting factors. Note that right choice of these two parameter sets may increase the rate of convergence, where on the other hand, a poor choice may cause the algorithm to diverge.

Apparently, this is an infinite choice tradeoff problem. However we can use a-priori knowledge and engineering intuition to choose the optimization parameters wisely (e.g., use ADOCS parameters as the initial guess). Moreover, using CONSOL-OPTCAD allows us to change our choice at each iteration during the design process, so using this feature one can develop a tradeoff strategy and even an optimal design "intuition". This idea, of using mixed optimal/classical techniques has already been used in the LQG/LTR [2], and in the symmetric root-locus [3] methods. In both methods the optimization step ensures a certain performance level (at least stability), while the other step allows the designer to improve the performance using classical considerations. These methods, and hopefully our approach as well, compensate for the loss of intuition that usually occurs when using pure optimal design techniques.

The organization of the report is as follows. In Section 2, we describe the completed parts of the research, including the helicopter model and the handling qualities requirements. In Section 3, we discuss in detail the two major problems that we faced, and their solutions. These problems are emphasized for two reasons. First, this information may be important for further works. Moreover, the proposed research outline described in [4] and [5] has been changed in order to solve these problems and others, such that we are now several months behind the original schedule. A short description of the present status is given in Section 4. A program and schedule for future

work are given in section 5, the final section.

Practically for the given specifications, one of the design goals is to ensure the internal stability of the closed-loop system. That is, the closed-loop system matrix A in a state-space representation of the closed-loop system is required to have all eigenvalues in the open left of the complex plane. This problem can be formulated as a non-smooth optimization by minimizing the largest real part of the eigenvalues of A. Typically, the process of minimization will tend to converge to a solution where many eigenvalues of A have the same real part. Moreover, it may not be possible to express the non-smooth objective function as the maximum of finitely many smooth functions and this difficulty presents a need in theoretical research. Progress has been made along this line by our sub-contractor at Georgia Tech. Among other things, a smooth reformulation of the problem and an algorithm are proposed such that a quadratic rate of convergence is often achieved. A report summarizing the results is attached in Appendix C. This algorithm will be eventually implemented in CONSOL-OPTCAD to simultaneously address other design specifications.

2 Completed parts

2.1 A model for the UH-60 in hover

The UH-60A (Black Hawk) helicopter in hover, was chosen as a benchmark example for this research. The comprehensive rotorcraft aerodynamic model (UM-GenHel) [6] requires much too much computer time for one simulation run to be useful for the purposes of this research. Thus we wrote a simplified model which can be used by the optimization package (CONSOL-OPTCAD). The simplified model represents the helicopter linear dynamics and aerodynamics, as well as the most important system nonlinearity (i.e., actuator saturation). The specific representation of the helicopter in hover is, in fact, a part of the design process setup. Therefor this model has been modified several times in order to obtain the system performance measures as well as to satisfy some computational limitations.

The final configuration is shown in Figure 2.1, including:

(i) $P(s)$ - Linearized UM-GenHel 11 states model (9 states for the 6 DOF fuselage dynamics and 2 states representing the rotor flapping motion).

(ii) *Actuator Model* - The swashplate actuators are modeled using standard saturation functions for displacement and rate limits and a 1^{st} order approximation for the actuator dynamics (see

Figure 2.2). Note that the displacement saturation is implemented on the actuator command as is standard [7].

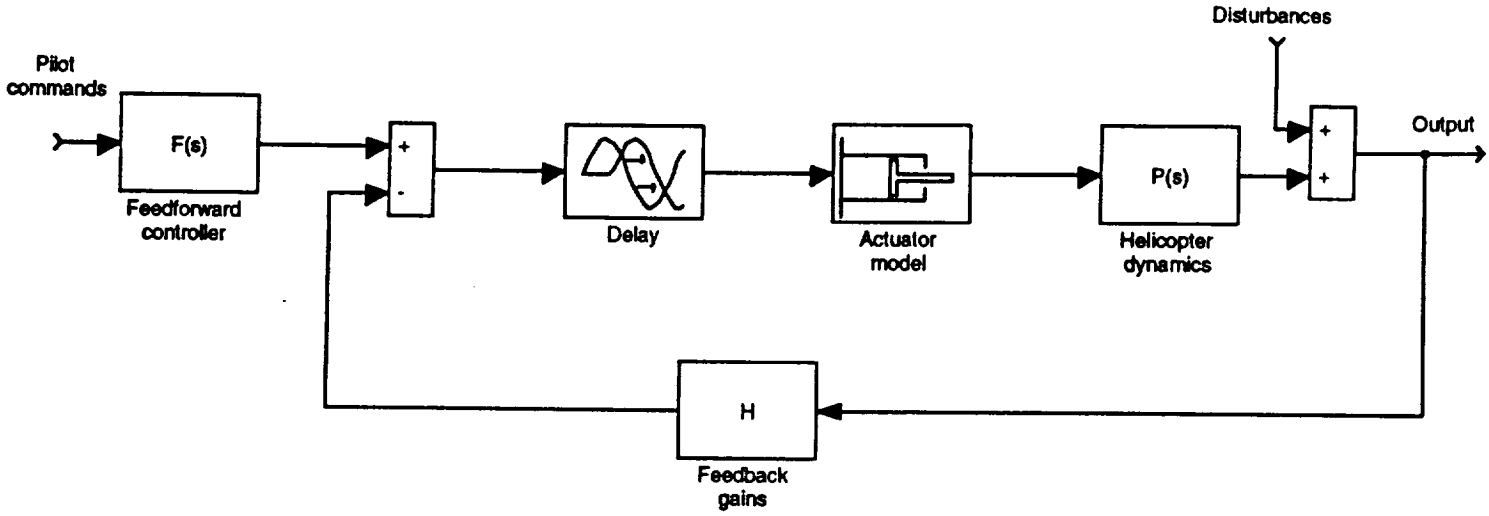

Figure 2.1 Block diagram of the simplified rotorcraft model

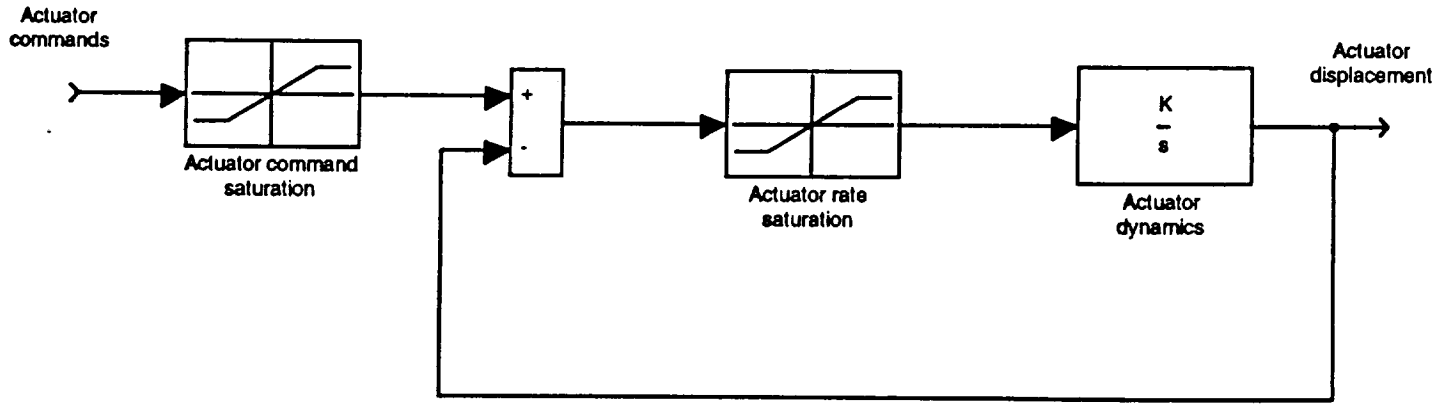

Figure 2.2 Block diagram of the actuator model

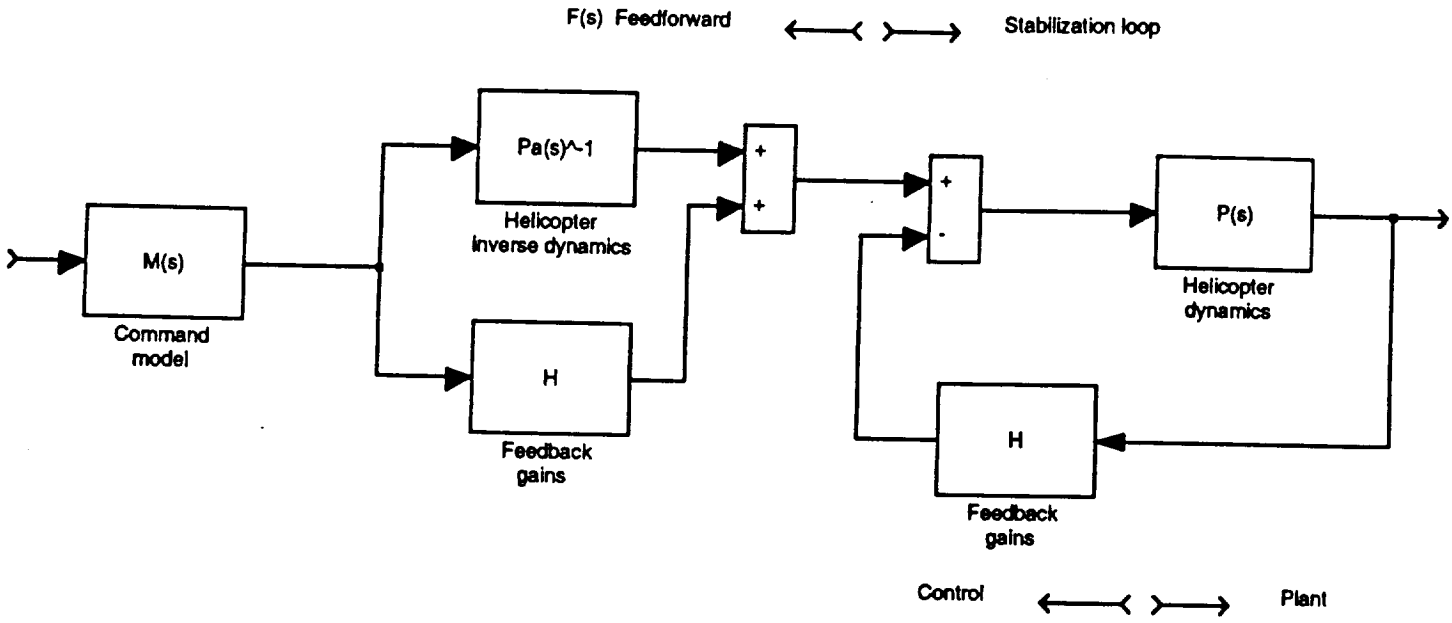

Figure 2.3 Detailed model of the feedforward portion of the rotorcraft controller

(iii) *Delay* - Pure delay which represents the overall system delay (computation delays, A/D and D/A delays, unmodeled dynamics etc.).

(iv) $H, F(s)$ - are ADOCS controllers. H is a constant gain, output feedback matrix. $F(s)$ is a decentralized feedforward dynamic controller obtained from the model following concept [8], based on a command model dynamics $M(s)$ and on a 1^{st} or 2^{nd} order approximation for the helicopter dynamics $P_a(s)$, see Figure 2.3. Note that if $P_a(s) = P(s)$ then the resultant closed-loop transfer function is $M(s)$.

The overall closed-loop system is controlled by the pilot using four input commands $\delta = (\delta_\phi, \delta_\theta, \delta_\psi, \delta_c)$ representing respectively the lateral, longitudinal, main rotor collective, and tail rotor collective cockpit commands. The output is $y = (u, v, w, p, q, r, \phi, \theta, \psi)$, which stands for the longitudinal velocity (u), lateral velocity (v), vertical velocity (w), roll rate (p), pitch rate (q), yaw rate (r), roll angle (ϕ), pitch angle (θ), and yaw angle (ψ). The disturbance vector $d = (d_\phi, d_\theta, d_\psi)$ allows us to simulate the helicopter angular rate changes caused by wind gusts.

In order to calculate efficiently all the desired performance measures, this model has two different versions. There is a continuous-time linear model, for small amplitude performance, where the actuator model $A(s)$ is a simple 1^{st} order model (no saturation) and the delay $D(s)$ is a

2^{nd} order Pade' approximation. There is also a discrete-time nonlinear model, for large amplitude performance, where all the continuous time parts (i.e., $P(s)$ and $F(s)$) are replaced by suitable ZOH equivalents. The nonlinear simulation is implemented by solving the difference equations recursively. For more details see MATLAB M-files, *init.m* and *simu.m* in Appendix A.

An important component of the initial task of this project was to verify the helicopter model. The linearized reduced order UM-GenHel model (11 states) was compared with the full UM-GenHel model (33 states) [6], and with the simplified 6 DOF model (9 states) [8]. In addition, the linear and the nonlinear closed-loop models were checked and compared using the "real" ADOCS parameters (i.e., feedback gains, and feedforward parameters). For further discussion of model verification see Paragraph 3.2.

2.2 Design specifications and their translation to mathematical functions

The following five specifications were identified as those that are essential to meet the ADS-33C (Aeronautical Design Standard) [9]. The helicopter is considered to hover ([9], Paragraph 3.3 - hover and low speed). The flight control system has an ACAH (Attitude Command Attitude Hold) response type. The standard to be met is performance LEVEL 1 for all MTEs (Mission Task Elements) excluding target acquisition and tracking for UCE (Usable Cue Environment) 2 and 3 (for detailed definitions see [9]).

These five requirements are naturally divided into two groups. The first two requirements relate to small-amplitude response and can be checked using linear models. The remainder are related to moderate-amplitude response and must be checked by nonlinear simulation.

In some cases the original requirements were changed slightly in order to achieve mathematical properties (i.e., smoothness) that simplify the optimization process. However, because of saturation the smoothness property is not always guaranteed.

2.2.1 Small-Amplitude Changes, Short-Term Response to Control Inputs

This modern frequency based criterion (bandwidth/phase-delay) replaces the traditional specifications which used limits based on time-delay and rise-time. The bandwidth/phase-delay criterion emphasizes features directly related to closure of the piloted loop, and it is a better metric than rise-time for the prediction of handling qualities for small-amplitude precision tracking tasks. It is clear that pilots are also sensitive to the shape of the phase curve at frequencies beyond the

bandwidth frequency. This shape is characterized by the phase-delay parameter [7]. Thus, the level regions for the short-term response requirement are defined in the bandwidth/phase-delay plane as shown in Figure 2.4. Actually, for small phase-delay systems this is a "pure bandwidth" criterion. Above a certain value of phase-delay (about 0.2 sec) it becomes a tradeoff between bandwidth and phase-delay (i.e., the pilot can tolerate higher phase-delay but then, in order to achieve the same performance level, he needs higher bandwidth).

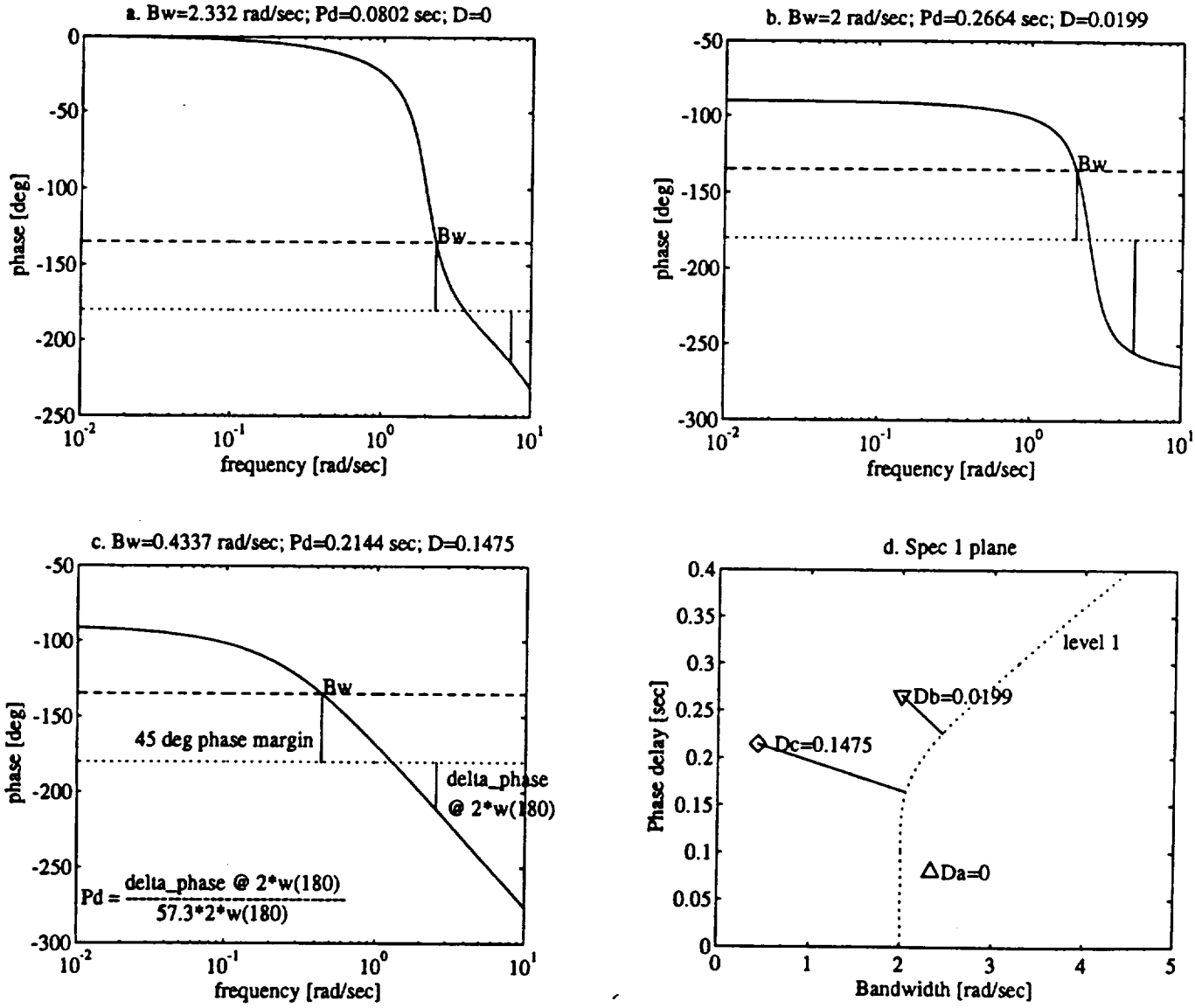

Figure 2.4 Spec 1 : Bandwidth vs. Phase-delay

(a-c) Bode phase plot, bandwidth and phase-delay of three examples. (d) Their D meassures in the Spec 1 plane.

Bandwidth and phase-delay are measured from a frequency response (Bode) plot of angular attitude response to cockpit controller input. Bandwidth and phase- delay, as defined in the specification [9], Paragraphs 3.3.2.1 (Pitch, Roll) 3.3.5.1 (Yaw), are referenced to the helicopter with all augmentation loops closed. Thus, they can not be simply calculated from the closed-loop design parameters. Usually, two bandwidth frequencies are measured: the frequency for 6 *db* gain margin ($\omega_{BW_{gain}}$), and the frequency for 45° of phase margin ($\omega_{BW_{phase}}$). For ACAH response types $\omega_{BW_{phase}}$ taken, since the nature of ACAH is such that the pilot does not have to close the attitude loop for stabilization purposes, so the gain margin problems are less apparent. Phase-delay is defined so that it represents all of the contribution to phase less than $-180°$, and is based on the observation that the phase curve tends to be linear.

In order to meet the LEVEL 1 (also true for other levels) requirement a two dimensional geometrical measure D (normalized quadratic distance) is defined, such that minimizing this measure implies better performance. The computation of this measure is done in three steps. First, the graphical level curve is converted into an analytic smooth function using a polynomial curve fitting algorithm. In order to achieve a univalent function the level curve is represented as $\omega_{BW} = f(\tau_d)$, i.e, the bandwidth frequency is a function of the phase-delay, which is a nondecreasing C^∞ function, Figure 2.4 d (the dotted curve). This calculation is done only once in the initialization routine (for more details see init.m in Appendix A). The other two steps are executed in each iteration. First, the bandwidth and phase-delay are computed, based on the above definitions, using an efficient search algorithm. Second, the D measure is calculated as the minimum normalized quadratic distance from the current (ω_{BW}, τ_d) point to the level curve. Moreover, measure D is calculated only for points which are not in the LEVEL 1 region (i.e., they are to the left of the dotted curve in Figure 2.4 d), and it is set to zero for any (ω_{BW}, τ_d) point within the LEVEL 1 region. This definition makes $D(\omega_{BW}, \tau_d)$ differentiable even on the boundary of the LEVEL 1 set (i.e., $\omega_{BW} = f(\tau_d)$), and gives an identical weight for any point in the desired set (LEVEL 1 region). The computation of D uses the fact that the level curve is a nondecreasing function, so it eliminates the need to evaluate the function $\omega_{BW} = f(\tau_d)$ at each $\omega_{BW} = f(\tau_d)$ (for more details see *d_bw_pd.m* in Appendix A).

2.2.2 Small-Amplitude Changes, Mid-Term Response to Control Inputs

This requirement is, in general, the complementary part of the short-term response requirements of Paragraph 2.1.1. The short-term criterion emphasizes features related to the high-frequency modes, whereas the mid-term influences mainly the low-frequency modes. Although, because of the particular definitions of both criteria, it is unavoidable that this requirement overlaps the short-term one. The mid-term requirement is specified for two different pilot operation modes: "Fully Attended Operations" - where all of the helicopter tasks can be accomplished with full pilot attention to aircraft control (e.g., other crew members handle the non-control tasks), and "Divided Attention Operations" - where the pilot should be able to relinquish control of the helicopter for short periods of time without encountering significant excursions. Consideration of divided attention (as done in this research) ensures that, at least practically, any flight control system which meets this requirement is BIBO stable. In fact a helicopter which meets the fully attended requirement can have unstable mid-term response (as with many present-day helicopters). For more information see [9], Paragraphs 3.3.2.2 (Pitch, Roll) 3.3.5.2 (Yaw).

The parameter that is used in this requirement is the damping ratio ζ. This parameter, which is well defined for second-order systems, can be interpreted in several different ways for higher order systems. Unfortunately, most of these interpretations lead to numerical algorithms which are either significant time consumers, or are not smooth enough, or both. For example, computing ζ as the logarithmic decrement of the two first peaks of the system step response is very time consuming. On the other hand, using eigenvalues, in order to find ζ as the ratio between the imaginary and the real part of the 2^{nd}order approximated model is, in general, not a smooth calculation. The computer time problem becomes critical because this algorithm has to be executed in a multi-iteration optimization process. The smoothness problem is sometimes even more critical since it may cause failure of the optimization process.

Usually, ζ defined only for stable systems. Thus practically, this requirement also implies stability. Indeed, the 2^{nd} order approximation is based on two parameters a and b $\left(g(s) = \frac{1}{s^2+as+b}\right)$ which completely define the system stability (i.e., $a > 0$, $b > 0$). Calculating ζ from a and b $\left(\text{i.e., } \zeta = \frac{a}{2\sqrt{b}}\right)$ implies that b must be positive (for numerical reasons, since ζ is a real number). Therefor, although it is not explicitly required, the optimization algorithm is forced to search for a solution only for $b > 0$ (to avoid numerical failure this requirement is translated to a "hard constraint", see [1]). In the future, in order to ensure stability, we will replace this constraint by

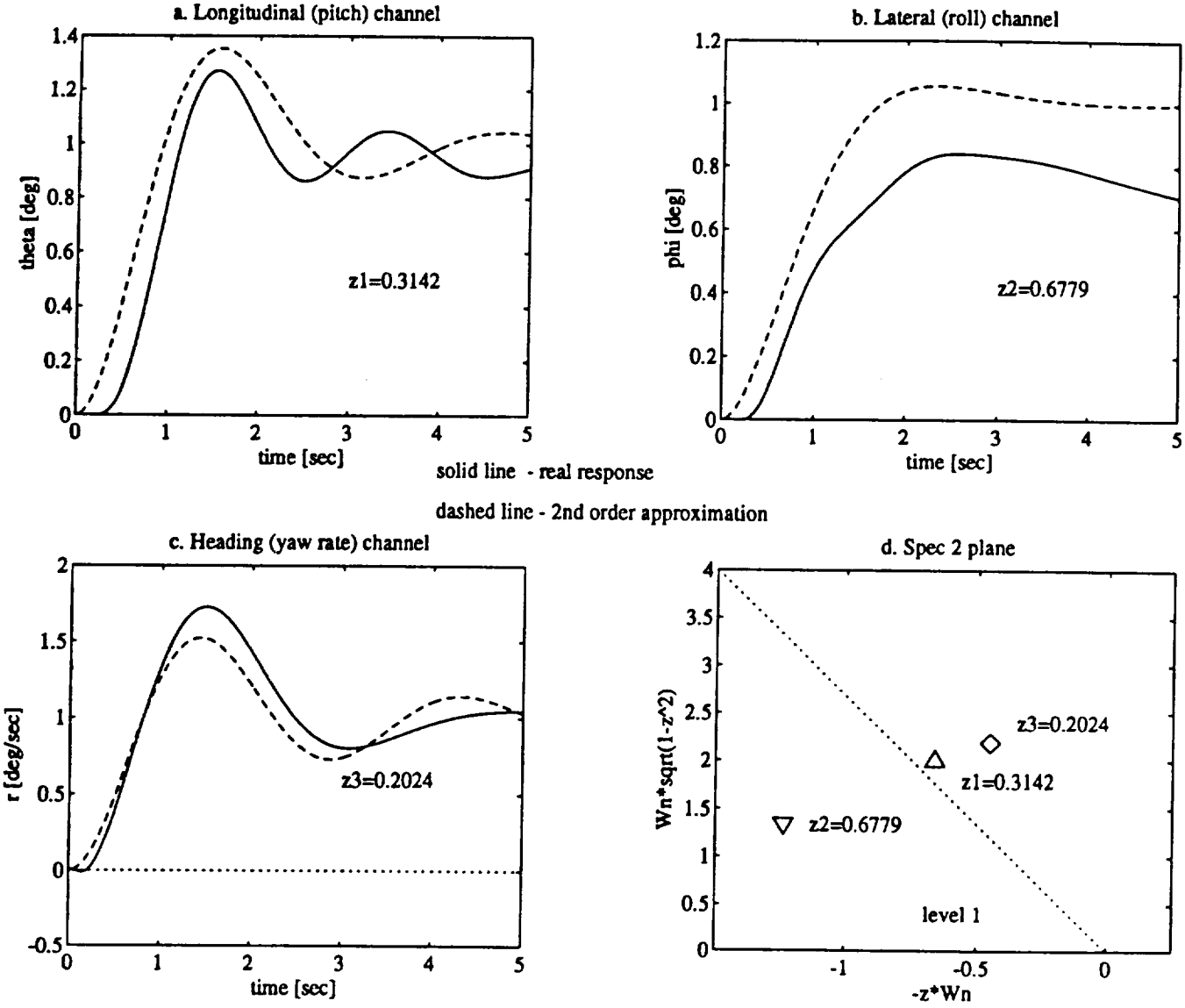

Figure 2.5 Spec 2 : Damping ratio

Responses for longitudinal (a), lateral (b), and heading (c) channels. (d) Their ζ meassures in the Spec 2 plane.

the algorithm described in Appendix C. Under this constraint, and by expanding the definition of ζ to hold also for negative values, the system stability becomes a function of ζ only (i.e., stable for $\zeta \geq 0$, and unstable for $\zeta < 0$).

During this work we have examined two ways for calculating ζ, both based on model reduction

techniques. First, we tried to use algebraic methods such as minimal realization using both a pole/zero cancellation method and a state elimination method. Unfortunately, these methods can not be applied for arbitrary models (i.e., the minimal size of the reduced model depends on the size of the full one). Thus usually they do not guarantee reduction to 2^{nd} order. In the current approach the 2^{nd} order approximation is obtained by using 2^{nd} order system identification. In this technique a second-order AR (auto-regressive) model is estimated using the LS (least-square) method. This approach needs only a few time points (about 20) and the LS algorithm ensures smoothness (for more details see *zeta.m* in Appendix A). Typical closed-loop, real and approximated, responses are shown in Figure 2.5.

2.2.3 Moderate-Amplitude Attitude Changes (Attitude Quickness)

Frequency domain based criteria (e.g., bandwidth) fail in the presence of strong nonlinearities such as saturation. Therefore, in order to check the helicopter performance during large maneuvers we have to define alternative criteria. Let α denote the nominal value of the angular position (ϕ, θ, ψ) for step response (i.e., α = step height), and let s_{pk} denote the peak value of the angular velocity (p, q, r). Then, for linear systems the ratio s_{pk}/α is directly related to the system bandwidth [7]. Using this criterion instead of one of the classical linear measures of bandwidth allows the

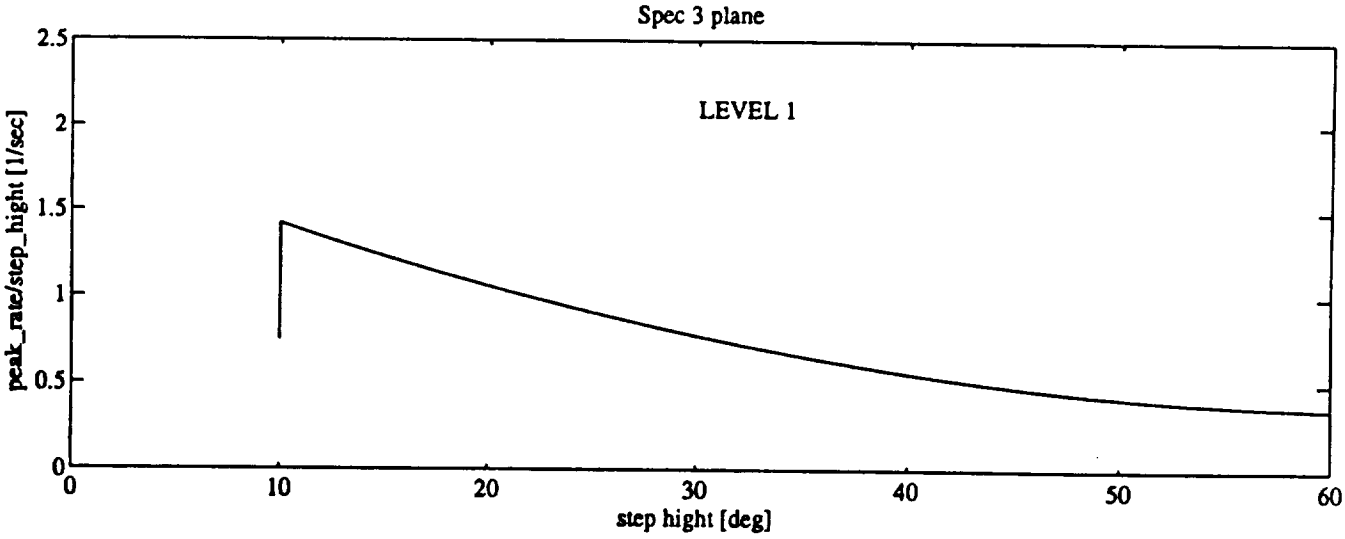

Figure 2.6 Spec 3 : Quickness ratio

Level 1 performance is achieved if the quickness ratio is above the curve shown.

specification to make the required bandwidth a decreasing function of the size of the maneuver,as shown in Figure 2.6. For nonlinear systems, especially with saturation nonlinearities, this concession is essential (i.e, it is unreasonable to require the same "bandwidth" for all input heights). In these cases it is not correct to interpret this ratio as a bandwidth, but it is better interpreted as a measure of agility (i.e., "quickness" ratio). For more information see [9], Paragraphs 3.3.3 (Pitch, Roll) 3.3.6 (Yaw).

The calculation of the quickness ratio s_{pk}/α is very simple. The $max(\cdot)$ operator is used over the sampled rate vector. It is then normalized by the input height. Note that using the $max(\cdot)$ operator in this case does not destroy smoothness because we try to maximize the quickness ratio and the combination max/max is smooth. However, smoothness may not hold due to the saturation. This test theoretically has to be checked for an infinite number of step inputs. Practically we check it for each channel for only three different inputs (for more details see *simu.m* in Appendix A).

2.2.4 Interaxis Coupling

Helicopters dynamics are naturally interaxis coupled. The following requirements relate to the two common helicopter interaxis couplings. These are the cross-coupling between pitch and roll (i.e., pitch/roll and roll/pitch), and yaw (rate) due to collective. These couplings are caused mostly by aerodynamic rotor moments and by the unsymmetric tail moments. Both couplings would adversely affect the pilot's ability to complete some high maneuvers tasks. One of the most important design objectives is to minimize these couplings (e.g., using closed-loop techniques such as: high-gain, LQR, etc.).

The decoupling requirement is mostly significant for high-maneuver responses. Thus, it is checked only for large amplitude step responses (The coupling measures are relative measures. Hence, in the case of linear systems small input amplitudes can be used as well). The natural measure is used for the cross coupling between pitch and roll (i.e., the ratio of peak off-axis response to desired response, θ_{pk}/ϕ_{des}, and ϕ_{pk}/θ_{des}). To avoid use of nondifferentable functions such as $max(\cdot)$ and $abs(\cdot)$, suitable upper and lower limits are defined to bound the response over the relevant time interval.

For the yaw-to-collective decoupling requirement a much more involved criterion is required. This criterion is designed to meet the pilot's needs during aggressive tasks. This criterion is a measure of not only the magnitude, but also the shape of the yaw rate response to a step collective

stick input. The shape of the yaw rate is taken into account by measuring the peak yaw rate $r1$ and value of yaw rate after 3 seconds. In case there is no peak in the time interval [0,3] the yaw rate at 1 second $r(1)$ is taken instead ([9], Paragraph 3.3.9). This may cause the measure to be not smooth with respect to the design parameters. It is clear that this can happen only if, during the optimization process, the damping ratio ζ of $\frac{r(s)}{\delta_c(s)}$ passes the value $\frac{1}{\sqrt{2}}$ (i.e., $\zeta(n) \leq \frac{1}{\sqrt{2}}$ and $\zeta(n+1) > \frac{1}{\sqrt{2}}$ or vice versa, where n is the iteration number). To avoid this singularity we take $r1 = r(1)$ regardless of the peak time (i.e., even if the peak occurs within the time interval [0,3]). In addition to the above requirement, it is also required that the maximum oscillation amplitude, following a step collective change is below a certain limit. From accumulated experience this limit has to have units [7], (i.e., it is not relative as in the pitch-roll case). All the information required for the coupling specs is taken from the largest input simulation of the quickness test (for more details see *simu.m* in Appendix A).

2.2.5 Wind Gust Response

The model-following concept allows the designer to increase the I/O bandwidth (theoretically unlimited) by changing the parameters of the desired model $M(s)$ only. Actually, this is the major advantage of using a model-following control, because in most cases the designer can not achieve

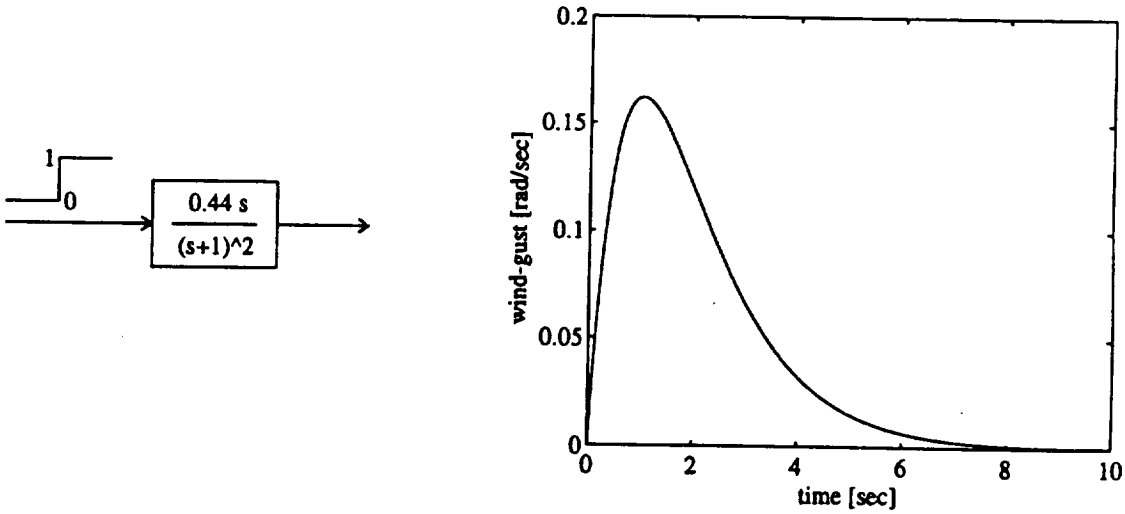

Figure 2.7 Spec 5 : Wind-gust model and wave form

the desired bandwidth due to closed-loop stability limitations (e.g., limited state feedback gains). However, this approach has some disadvantages. One of them is that the disturbance rejection requirement (system "stiffness") is no longer directly related to the overall system bandwidth (i.e., we can have a high-bandwidth, low- stiffness system). Therefore, the original wind-gust rejection criterion ([9], Paragraph 3.2.6) is not suitable for the ADOCS configuration. A new requirement is defined [10], by using an approximated gust model, where the gust peak value is chosen to fit the disturbance input point (the wind-gust is applied directly to the helicopter state equations). The wind-gust input wave form is shown in Figure 2.7, the detailed definition of the new requirement is presented in [10].

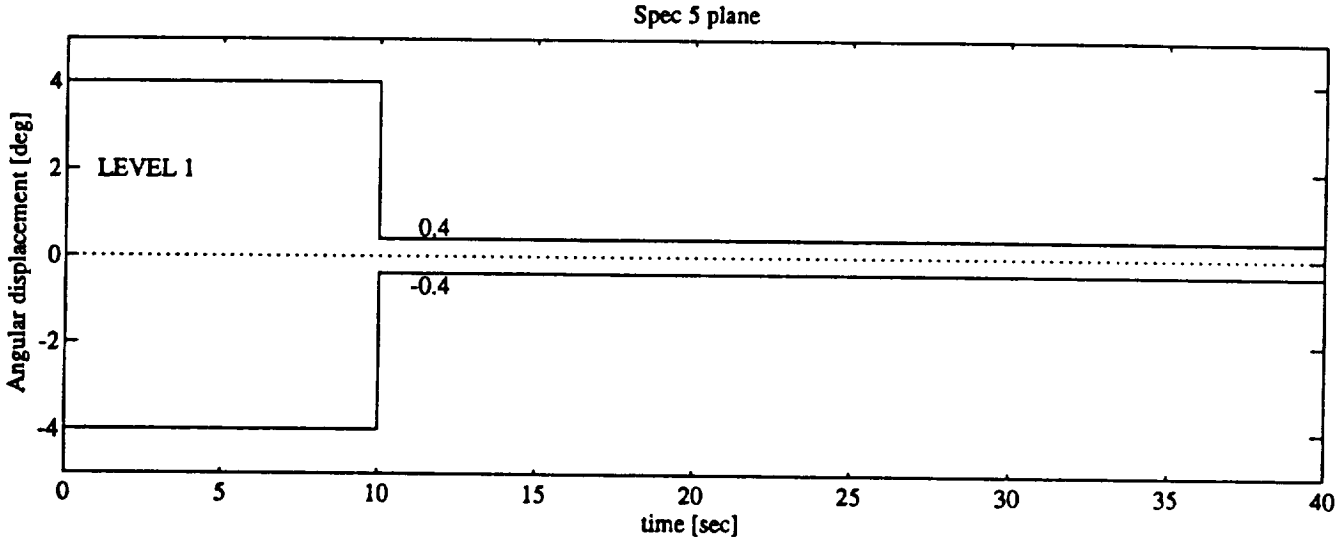

Figure 2.8 Spec 5 : Wind-gust rejection - required envelope

The wind gust response criterion is a two parameter criterion. Following precisely the requirements for the ACAH response type [9] leads to a pulse- input settling-time criterion. The first parameter, settling-time $t_s(a)$ is defined by the condition that the absolute value of the response $|y(t)| \leq a \; \forall \; t \geq t_s(a)$, where a is 10% of the response peak value. The second parameter is the peak value itself. Practically, settling-time is obtained using a discrete-time search algorithm, which may cause $t_s(a)$ to be not smooth with respect to the design parameters. Moreover, in order to obtain the peak value we have to use the $max(\cdot)$ operator, and since the optimization algorithm tries to minimize this peak it may cause some smoothness difficulties. To avoid this possibility it is highly recommended that one uses functional constraints. Therefore, the ACAH response

type requirement was slightly changed such that the angular position following a wind-gust input should lie between the two curves of Figure 2.8.

Remark: This test has to be simulated for 40 seconds. Thus it becomes the biggest time consumer of the simulation. One might think that in order to save simulation time, this test can be replaced by a mixed time domain (for the peak) and frequency domain (for $t_s(a)$) test. However, it turns out that then the problem constraints can not be functional constraints (smoothness !) and they must be more conservative in order to guarantee the same performance level.

2.3 Optimization set-up

The computerized optimization set-up includes 3 groups of files:

(i) Optimization definitions (see Appendix B, and [1]):

• *ADOCS* - CONSOL-OPTCAD Problem Description File (PDF), contains definitions for the design parameters, their initial values, and their limits (if there are any).

• *SPEC#.$* - Spec files where *# = 1,2,3,4,5;* stands for the specific spec (i.e., 1 - bandwidth vs. phase-delay, 2 - damping ratio, etc.), and $ = *pit* (pitch),*rol* (roll), *yaw*; stands for the specific controlled channel. These files include the problem constraints (types, values, and weights). These values can be changed at each iteration.

In the first stage of the optimization ("convert"), all the above files are included to one CONSOL-OPTCAD executable file.

(ii) Model and performance (see Appendix A):

• *INIT.M* - Initialization file, defines the open-loop parameters (helicopter linearized dynamics, actuator saturations, overall time delay, wind-gust wave form, etc.). This file is executed once, at the begining of each optimization run. The initial parameters can be changed (if required) after each iteration.

• *SIMU.M* - Contains closed-loop parameters including the design parameters, and the calculation of the performances measures. This file is executed at each iteration.

• **.M* - Series of function files, containing algorithms which have to be executed several times at each iteration (e.g., for the discrete recursive computation).

(iii) Monitoring tools:

This group of files allows the user to monitor the optimization process at any time, even if the process is running in the background. These tools were developed for two reasons.

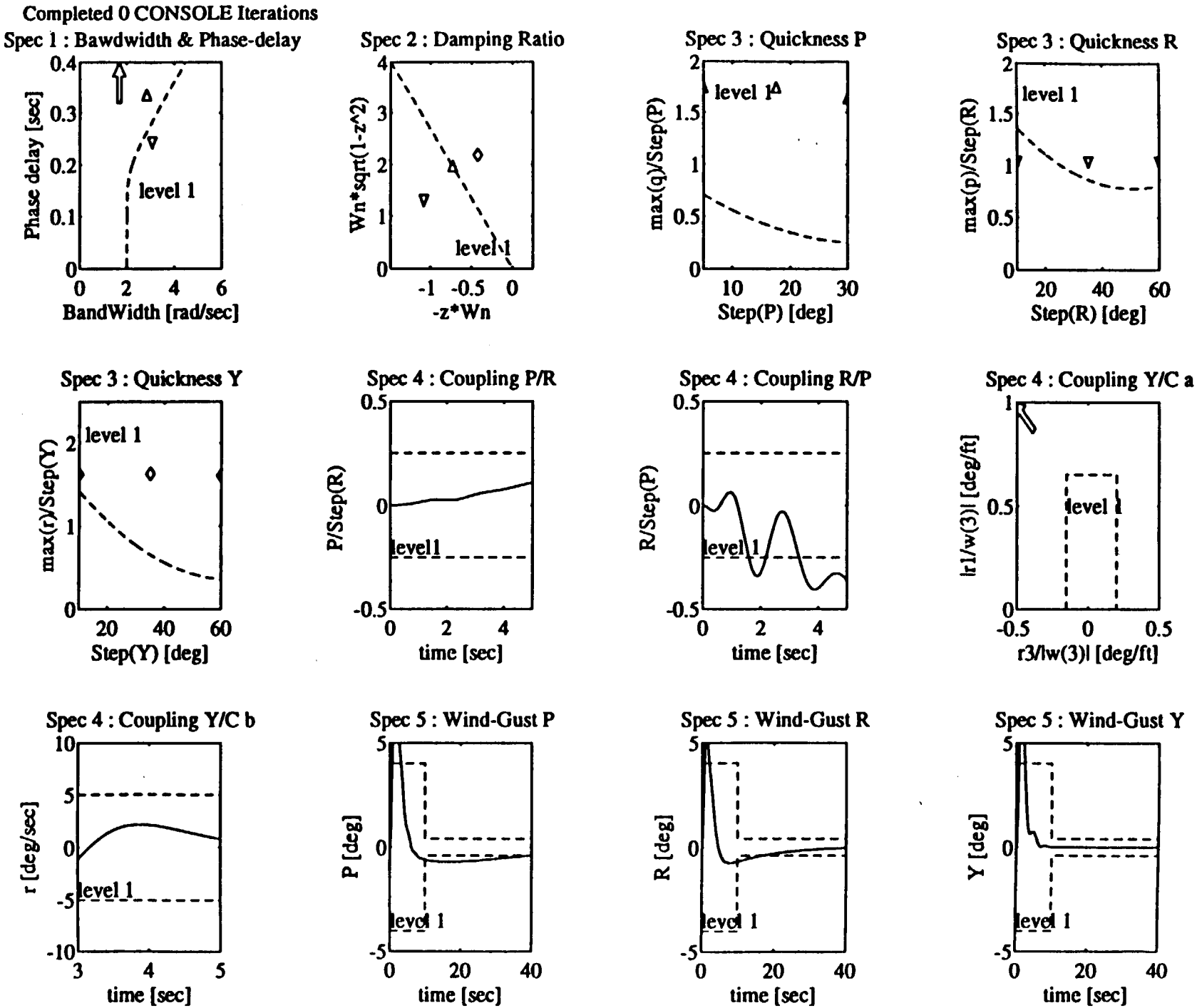

Figure 2.9 Performances map (initial)

First, the present version of CONSOL-OPTCAD has limited graphical outputs (a new version including a strong graphical interface is currently being developed). Moreover, it is desirable to display all the performance measures at their specific (not standard) planes, preferably in one screen (page), as shown in Figure 2.9. This feature gives the designer the ability to see the present performance map, at any time, just by pushing one button.

3 Major Problems

During this research we faced many problems (mathematical problems, numerical problems, technical difficulties, computer bugs, etc.). Two of these problems had substantial influence on the research outline and rate, in particular the first one (3.1), for which we spent several months to find the present (probably not the optimal) solution.

3.1 Simulation running time

In retrospect, we can say that this problem is a direct result of two poor choices made at the beginning of this research: choosing MATLAB as the main simulation tool and the linear continuous-time model as a starting test case. However, in this early stage, these choices seemed to be the right ones. At the beginning, we used a simplified test case, in order to develop and to check some of the performance measures. Then, MATLAB was a natural choice, mainly because of its powerful linear algebra tools. Although, MATLAB is an interpreter (i.e., its code can not be compiled to produce an executable file) the running time per one CONSOL-OPTCAD iteration was reasonable (several minutes). However, when we started to complicate the model (i.e., adding the saturation nonlinearity) the running time increased dramatically (more than 10 hours per one iteration). The main source for this unacceptable simulation duration was the nonlinear simulation. This was initially implemented by using numerical integration using the Runge-Kutta 4^{th} or 5^{th} order variable step size algorithm (see MATLAB M-file, *ode45.m*). This method is relatively slow, in particular in MATLAB, where it is implemented very inefficiently.

At the first step, we tried to speed up the algorithm by converting the M-file to a C MEX-file (compiled subroutine) and by improving the original algorithm interface (see MATLAB MEX-file, *ode45m.c*), but the minimum simulation time that we achieved was about 10 hours per optimization iteration. The second step was to replace the continuous time model with a discrete time one, and

then the numerical integration was replaced by an efficient recursive calculation. Moreover, this change saved 8 states used for the time delay Pade' approximation (in discrete time, time delay has a very simple implementation). Finally, after some more simplifications (e.g., reduce rotor dynamics to a 2^{nd} order flapping model, replace wind-gust model with its a-priori calculated wave form, etc.), we ended with 30 - 40 minutes per iteration. During the research, we have considered switching to an independent simulation code written in C or FORTRAN, but it now seems that that is no longer worthwhile, because all our working environment is based on MATLAB.

This problem affected the research by consuming the time needed for solution searching. Moreover, it implies that in the next research steps, in order to get results in reasonable time, we must use a smart computer run policy as discussed in Section 4.

3.2 A bug in UM-GenHel code

At the beginning of the model verification (August 1992), we realized that the given linearized model gave wrong results. First, we verified that closing the loop with the real ADOCS parameters [11] produced an unstable closed-loop system. After several months (December 1992) of testing and comparing the linearized model and the UM-GenHel/ADOCS interfaces without any improvement the nonlinear UM-GenHel simulation was compared with previous simulation results to find that there was a bug in the UM-GenHel code. It is not clear why or when this bug appeared ? However, it is clear that the wrong model was used at least in one research [12]. Finally, a new version was installed and, after some adjustments were made (February 1993), we obtained the present linearized model.

4 Present status

The main activity now is the tradeoff runs. We have already finished the first set of nominal runs starting with an arbitrary initial guess, (e.g., ADOCS parameters [11]), and with a fixed arbitrary set of constraint weights for a fixed number of iterations. However, since we chose all the optimization parameters arbitrarily, and we did not apply any tradeoff strategy, the final solution is an arbitrary one, and hence it is not the "optimal" one. The final and the initial performance maps for this case are shown in Figure 2.9 and Figure 4.1 respectively.

Unfortunately, as we already mentioned, this design problem is not convex and maynot be

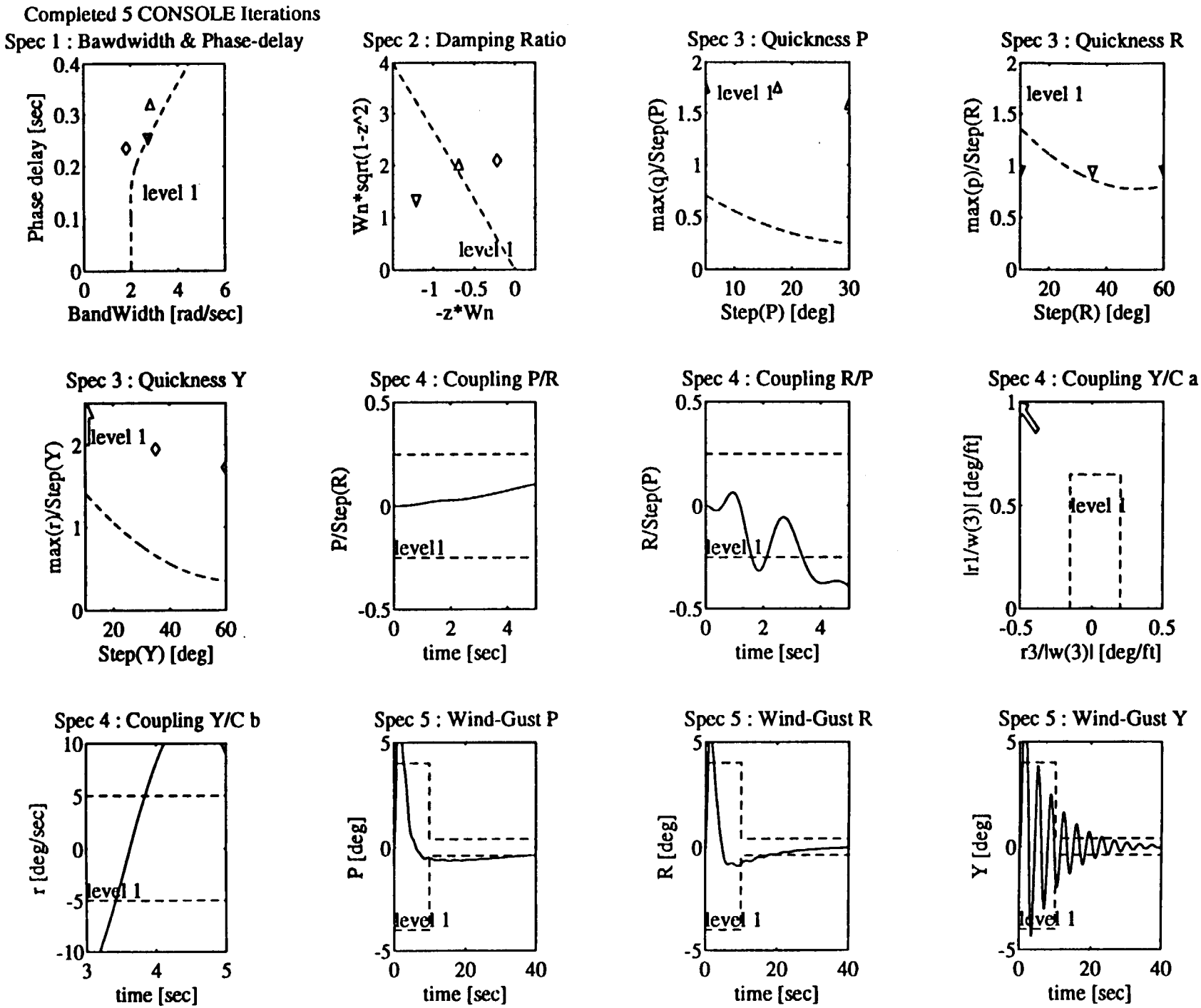

Figure 4.1 Performances map (final)

Spec No.	Measure	Pitch	Roll	Yaw
1.	*distance*	(1)	(2)	(3)
2.	ω_n^2	[4]	[6]	[8]
	ζ	(5)	(7)	(9)
3.	*small*	(10)	(13)	(16)
	medium	(11)	(14)	(17)
	large	(12)	(15)	(18)
4.	*coupling*	$< 1,2 >$	$< 3,4 >$	$< 5,6 >$ (19,20) (21,22)
5.	*peak*	$< 7,8 >$	$< 11,12 >$	$< 15,16 >$
	$t_s(a)$	$< 9,10 >$	$< 13,14 >$	$< 17,18 >$

Table 4.1 Problem constraints

smooth everywhere. Therefore, in order to help the optimization process to converge to an acceptable solution we must adjust the design degrees of freedom: initial guess, and constraint weights. At the first step (currently developed) we are going to use a tradeoff strategy only for the constraints weights. This may give us the ability to control the local solution (the global solution strongly depends on the initial guess). For this purpose we have a set of design DOFs, which are the "soft constraints" shown in Table 4.1, where each (#) or $< \# >$ stands for one design DOF (total 26). Note that (#) (or (#, #)) is a CONSOL-OPTCAD "soft constraint" (or a pair of "soft constraints"), [#] is a "hard constraint", which has no weight(i.e., it is not a design DOF), and $< \# >$ (or $< \#, \# >$) is a "functional (soft) constraint" (or pair of "functional constraints"). For more details see spec files in Appendix B, and [1]).

The "good values" (GV), which are exactly the spec requirements, remain fixed during the design process and the constraints weights (CW) are directly related to the "bad values" (BV). Practically, in the j^{th} CONSOL-OPTCAD iteration we can change the "bad values" using the factors $\beta_{i,j}$ such that:

$BV_{i,j} = (1 - 1/\beta_{i,j})GV + 1/\beta_{i,j}BV_{i,j-1}; \;\; i = 1,2,\ldots,26,$

which implies:

$CW_{i,j} = \beta_{i,j} CW_{i,j-1}$

This approach using a normalized multiplicative weighting factor, easily allows us to interfere at each CONSOL-OPTCAD iteration, and "push" the algorithm toward the right direction.

5 Proposed work

We have completed approximately half of the tasks we originally scheduled during the first year. The only real change to our protocol is the use of somewhat more simplified model in the design process than was originally planned. We do not believe this will introduce any changes in the rest of the project. Our primary goal remains the development of CONSOL-OPTCAD as a tool for the design of control systems for advanced rotorcraft. It is very important to understand that we are developing a tool, not an algorithm. A human designer plays the crucial role in designing the controller. As with many sophisticated and complex tool it is not possible to simply hand our customization of CONSOL-OPTCAD to a designer and say, "Go design a controller". There is a great deal to learn about how to set-up the design problem and about how to force CONSOL-OPTCAD to arrive at a good design. This is the major focus of our proposed research for the coming year.

As last year, there are two major issues, controller structure and controller parameters. CONSOL-OPTCAD is fundamentally a parameter optimizer. Thus, it must be given the structure of the controller. There are several ways to do this. We will first use the ADOCS structure and optimize its parameters, as we had originally planned to do last year. We view this as a form of inverse problem. That is, as a first step we will try to force CONSOL-OPTCAD to converge to the ADOCS design. We know this is not a perfect design but it is a very useful way to learn how the parameters with which the designer influences the design (the good and bad values of the mathematical versions of the specs primarily) influence both the evolution and the result of the CONSOL-OPTCAD computations.

Once we have done this we would like to experiment with the choice of controller structures. We will first use our intuition and conversations with various rotorcraft experts to produce some other controller structure. We will then use CONSOL-OPTCAD to choose good parameters for those controllers , study their sensitivity to changes in the controller parameters, and evaluate their robustness. If time permits we would then also like to try an LQR/LTR design using the methodology outlined outlined in the introduction.

We propose to follow the schedule given below.

1. Design studies using the ADOCS structure (completed by August 1993)
   - 1.1. Study performance specification tradeoffs.
   - 1.2. Study controller sensitivity to specification.
   - 1.3. Study controller robustness.
   - 1.4. Solve the inverse problem - i.e., force CONSOL-OPTCAD to converge to the ADOCS parameter values.
2. Design studies using other prespecified structures (completed by Jan. 1994)
   - 2.1 - 2.4. Repeat the studies done for the ADOCS structure.
3. Design studies using LQR/LTR (completed by March 1994)
   - 3.1 - 3.4. Repeat the studies done for the previous structures.

Remark: The third task will only be done if time permits. We believe it would be very useful to demonstrate the possibility of combining CONSOL-OPTCAD with different optimal control methods but it is not necessary for the success of the proposed project.

6. References

[1] M. K. H. Fan, A. L. Tits, J. L. Zhou, L. Wang and J. Koninckx, "CONSOL - OPTCAD," University of Maryland at College Park, Systems Research Center , Tech. Report TR 87-212r2a, Aug., 1991.

[2] G.Stein and M. Athans, "The LQG/LTR Procedure for Multivariable Feedback Control Design," *IEEE Trans. AC* , Vol. AC-32, pp. 105–114, February, 1987.

[3] T. Kailath, *Linear Systems.* Prentice - Hall, 1980.

[4] W. S. Levine and J. Barlow, *Research Proposal - Techniques for Designing Rotorcraft Control Systems.* University of Maryland, Systems Research Center, July, 1991.

[5] G. Yudilevitch, Research Plan , June, 1992.

[6] F. D. Kim, "Formulation and Validation of High-Order Mathematical Models of Helicopter Flight Dynamics," Ph.D. Thesis, University of Maryland, College Park, Department of Aerospace Engineering, 1991.

[7] R. H. Hoh, D. G. Mitchell, B. L. Aponso, D. L. Key and C. L. Blanken, "Background Information and User's Guide for Handling Qualities Requirements for Military Rotorcraft," US Army ASC, USAAVSCOM TR 89-A-8, Dec., 1989.

[8] M. B. Tischler, "Digital Control of Highly Augmented Combat Rotorcraft," NASA, US Army ASC, NASA Tech. Memo. 88346, USAAVSCOM TR 87-A-5, May, 1987.

[9] US Army ASC, "Handling Qualities Requirements for Military Rotorcraft," Aeronautical Design Standard, ADS-33C, Aug., 1989.

[10] M. Tischler, Sample Disturbance Response Specification, Sept., 1990.

[11] M. B. Tischler, J. W. Fletcher, P. M. Patrick, M. Morris and G. E. Tucker, "Flying Analysis and Flight Evaluation of a Highly Augmented Combat Rotorcraft," *J. of Guidance, Control, and Dynamics*, Vol. 14, No. 5, pp. 954–963, Oct., 1991.

[12] N-K. Tsing, "Computer-Based Techniques for Control System Design, with Application to Rotorcraft Control," Ph.D. Thesis, University of Maryland at College Park, Department of Electrical Engineering, 1992.

```
% ************************************************************************
%   System Initialization File for ADOCS
%   init.m CONSOL-MATLAB file
%   Continous-Linear Model for specs: 1 (Bandwidth-Phase delay) & 2 (damping)
%   Discrete-Nonlinear Model for specs: 3 (Quickness), 4 (coupling),
%   & 5 (Wind-gust)
%   By : Gil Yudilevitch
%   Last update  :   02/23/93
% ************************************************************************

% System Block Diagram

%                                                                  Gd
%                                                                  |
%  R  ,---,Rm  E  ,---,Ed ,----,Rs     ,----,Us ,-----,Xs  ,---,Yp V+  Y
% --->| F |-->O-->| D |-->| Sc |-->O-->| Sr |-->| K/s |-+->| P |-->O-+-->
%     '---' + ^-  '---'   '----' + ^-  '----'   '-----' |  '---' +   |
%             |                   |                     |            |
%             |      ,---,        '---------------------'            |
%             '------| H |<-----------------------------------------'
%                    '---'

%  GENERAL DEFINITIONS

dt = 1/30;            % Sampling rate 30 Hz
tf = 5; nt=tf/dt+1;  % Command Type Time Axis
t1=1/dt+1; t3=3/dt+1;
t =0:dt :tf;
tfg=40; tg=0:dt:tfg; % Disturbance Type Time Axis
tg10=find(tg > 10);
ntg=tfg/dt+1;

%  MODEL COMPONENTS
%  ----------------
%  1. Helicopter 6 DOF Dynamics + Rotor Aerodynamics
%     Linear Continuous Time Model

%  # of states    = 11 ( 9 for 6 dof dynamics + 2 for rotor aerodynamics )
%  # of inputs    = 4
%  # of outputs   = 7  ( 4 for the linear case --> C 4 x 11 )

%                                          State     Output   Input
%                                          -----     ------   -----
%  u      - linear velocity along X axis      1
%  v      - linear velocity along Y axis      2
%  w      - linear velocity along Z axis      3       * 5        4  delta_c
%  p      - angular velocity about X axis     4       * 6
%  q      - angular velocity about Y axis     5       * 7
%  r      - angular velocity about Z axis     6         4
%  phi    - angle about X axis                7         2        2  delta_phi
%  theta  - angle about Y axis                8         1        1  delta_theta
%  psi    - angle about Z axis                9         3        3  delta_psi
%  b1c    - longitudinal flap  (-a1s)        10
%  b1s    - lateral flap       (-b1s)        11
%                                                    * - used for NL only

B10to4 = zeros(10,4);          % Trans. 10 (UMGENHEL) --> 4 inputs
B10to4(1,2) = 1; B10to4(2,1) = -1; B10to4(4,3) = 1; B10to4(3,4) = 1;
%                              ^ the 1st channel in Mark's model is negative
%                              | see the oposite sign in the relevant feedback
%                              | gains (Ktet & Kq)

% CONTINUOUS TIME
% ---------------
```

```
load A11 -ascii; load B11 -ascii;                % Load A,B UMGENHEL matrices

Ap = A11;

Bp = B11*B10to4;

Cp = [ 0 0 0 0 0 0 0 1 0 0 0
       0 0 0 0 0 0 0 1 0 0 0
       0 0 0 0 0 0 0 0 1 0 0
       0 0 0 0 0 1 0 0 0 0 0];

Dp = zeros(4,4);

% DISCRETE TIME
% -------------

[Apd,Bpd] = c2d(Ap,Bp,dt);       % digital ZOH equi
[Apd,Bgd] = c2d(Ap,eye(11),dt);  % digital ZOH equi

% Pure Delay 2_nd Order Pade Approximation (for the
% ----------------------------------------
%
% Total Delay was taken equal for all channels. Tot
% from Mark's AIAA paper is 223 ms ==> 200 ms "pure
% Time-Constant (see later in SP-Actuator Model)

Td = [0.2;0.2;0.2;0.2];
[Ad1,Bd1,Cd1,Dd1] = pade(Td(1),2);
[Ad2,Bd2,Cd2,Dd2] = pade(Td(2),2);
[Ad3,Bd3,Cd3,Dd3] = pade(Td(3),2);
[Ad4,Bd4,Cd4,Dd4] = pade(Td(4),2);
[Ad,Bd,Cd,Dd]=append(Ad1,Bd1,Cd1,Dd1,Ad2,Bd2,Cd2,Dd
[Ad,Bd,Cd,Dd]=append(Ad3,Bd3,Cd3,Dd3,Ad,Bd,Cd,Dd);
[Ad,Bd,Cd,Dd]=append(Ad4,Bd4,Cd4,Dd4,Ad,Bd,Cd,Dd);
Ndel = round(Td/dt);  % # of delayed sampling inter

% Swashplate-Actuators Model
% --------------------------

% LINEAR PART :-

K   =  1/0.025*ones(1,4);              % 1/(Time Constant) equal actuators
[as1,bs1,cs1,ds1] = tf2ss([0,K(1)],[1,K(1)]);
[as2,bs2,cs2,ds2] = tf2ss([0,K(2)],[1,K(2)]);
[as3,bs3,cs3,ds3] = tf2ss([0,K(3)],[1,K(3)]);
[as4,bs4,cs4,ds4] = tf2ss([0,K(4)],[1,K(4)]);
[As,Bs,Cs,Ds] = append(as1,bs1,cs1,ds1,as2,bs2,cs2,ds2);
[As,Bs,Cs,Ds] = append(As,Bs,Cs,Ds,as3,bs3,cs3,ds3);
[As,Bs,Cs,Ds] = append(As,Bs,Cs,Ds,as4,bs4,cs4,ds4);

% NONLINEAR SATURATIONS :-

% *** All data is given at cockpit units (i.e., in's of stick or pedal)
% *** The trim (hover) conditions set to be zero (see table below)
% *** The swashplate mechanizem ("limits coupling") is NOT taken in acount

% Command Limits Table, Ref. Sikorsky SER 70452 p 6.16 Fig. 6.3.1 (Celi Oct 92)
% ..........................................................................
% channel   |     Pitch     |      Roll     |       Yaw       |  Collective  |
% actual    |-5.0 +0.2 +5.0|-5.0 -0.5 +5.0|-2.69 +0.85 +2.69| 0.0 +5.0 +10.0|
% simulation|-5.2  0.0 +4.8|-4.5  0.0 +5.5|-3.54  0.00 +1.84|-5.0  0.0 + 5.0|

Csl = [ -5.2 -4.5 -3.54  -5];   % Actuator Strok (Commands) Lower Limits
```

```
Csu = [  4.8  5.5  1.84   5];   % Actuator Strok (Commands) Upper Limits
Rsl = [-10  -10   -5.38 -10];   % Rate Lower Limits
Rsu = [ 10   10    5.38  10];   % Rate Upper Limits

% Linear Continuous-Time Open-Loop
% --------------------------------

[Ao,Bo,Co,Do] = series(Ad,Bd,Cd,Dd,As,Bs,Cs,Ds);
[Ao,Bo,Co,Do] = series(Ao,Bo,Co,Do,Ap,Bp,Cp,Dp);

% Closed-Loop (Design) Parameters, Initial Values
% *** ONLY FOR INIT.M INDEPENDENT RUN ***
% ----------------------------------------------

%Kq     = -16.0;  Kp    =   6.0;  Kr    =   7.2; % initial design
%Ktet   = -34.0;  Kphi  =  20.0;  Kpsi  =   7.7; % parameters

Kq     = - 6.8;  Kp    =   1.3;  Kr    =   4.0; % final design
Ktet   = -10.4;  Kphi  =   8.4;  Kpsi  =   7.6; % parameters

Mtet   =    2.0;  Mphi  =   2.5;  Mpsi  =   2.0;

FGtet = -0.329;  FGphi = 1.312;  FGpsi = 0.715;
FTtet =  0.519;  FTphi = 3.348;  FTpsi = 0.288;

% Time Response Inputs
% --------------------

DtR  = pi/180;
Stet = DtR*[ 5,17.5,30];
Sphi = DtR*[10,35  ,60];
Spsi = DtR*[10,35  ,60];
Scol = 5;

% Wind-Gust Time History
% ----------------------

Gust = step([0 0.44 0],[1 2 1],tg);

% Level 1 curve for spec 1
% ------------------------

Ts = [0.000 0.025 0.050 0.075 0.100 0.125 0.150 0.175 0.200 ...
      0.225 0.250 0.275 0.300 0.325 0.350 0.375 0.400];
Ws = [2.005 2.005 2.005 2.005 2.005 2.005 2.005 2.105 2.235 ...
      2.405 2.685 3.005 3.275 3.505 3.905 4.155 4.455];
PL1=polyfit(Ts,Ws,5);

% **************   end of init.m   ********************************************


% ****************************************************************************
%   System Simulation File for ADOCS
%   simu.m CONSOL-MATLAB file
%   Continous-Linear Model for specs: 1 (Bandwidth-Phase delay) & 2 (damping)
%   Discrete-Nonlinear Model for specs: 3 (Quickness), 4 (coupling),
%   & 5 (Wind-gust)
%   By : Gil Yudilevitch
%   Last updated  :  03/15/93
% ****************************************************************************
%----------------------------------------------------------------------------
diary err.msg;

% Feedforward (including Model Following & "Inverse Closed-Loop")
% ---------------------------------------------------------------

% Continuous Time
% ---------------

[Af_tet,Bf_tet,Cf_tet,Df_tet] = ...
tf2ss(Mtet^2*[1/FGtet Kq+1/FGtet/FTtet Ktet],[1 2*Mtet Mtet^2]);
[Af_phi,Bf_phi,Cf_phi,Df_phi] = ...
tf2ss(Mphi^2*[1/FGphi Kp+1/FGphi/FTphi Kphi],[1 2*Mphi Mphi^2]);
[Af_psi,Bf_psi,Cf_psi,Df_psi] = ...
tf2ss(  Mpsi*[1/FGpsi Kr+1/FGpsi/FTpsi Kpsi],[1   Mpsi      0]);
Af_c = 0; Bf_c = 0; Cf_c = 0; Df_c =1;
[Af,Bf,Cf,Df]=append(Af_tet,Bf_tet,Cf_tet,Df_tet,Af_phi,Bf_phi,Cf_phi,Df_phi);
[Af,Bf,Cf,Df]=append(Af,Bf,Cf,Df,Af_psi,Bf_psi,Cf_psi,Df_psi);
[Af,Bf,Cf,Df]=append(Af,Bf,Cf,Df,Af_c,Bf_c,Cf_c,Df_c);

% Discrete Time
% -------------

[Afd,Bfd] = c2d(Af,Bf,dt);

% Stabilization Feedback Gains Matrix
% -----------------------------------

H = [ ...
    0 0 0  0 Kq  0    0 Ktet    0 0 0
    0 0 0 Kp  0  0 Kphi    0    0 0 0
    0 0 0  0  0 Kr    0    0 Kpsi 0 0
    0 0 0  0  0  0    0    0    0 0 0];

% Linear Closed-Loop
% ------------------
% States order:
%  1 :  7 - Feedforward
%  8 : 15 - Delay (4 x 2nd order Pade apprx.)
% 16 : 19 - Swashplate actuators
% 20 : 30 - 6 DOF + rotor (outputs: theta, phi, psi, r)

Ac = Ao-Bo*[zeros(4,12),H];
[A,B,C,D] = series(Af,Bf,Cf,Df,Ac,Bo,Co,Do);

% Nonlinear Closed-Loop (Step Responses)
% --------------------
% Command Response | States order:
%  1 :  7 - Feedforward
%  8 : 11 - Swashplate actuators
% 12 : 22 - 6 DOF + rotor (outputs: theta, phi, psi, p, q, r, w)

Nsamp=round(tf/dt+1);

% Pitch input
% -----------

nR=1;
aR=Stet(1); drsc; X11 = X;
aR=Stet(2); drsc; X12 = X;
aR=Stet(3); drsc; X13 = X;

% Roll input
% ----------

nR=2;
aR=Sphi(1); drsc; X21 = X;
aR=Sphi(2); drsc; X22 = X;
```

```
aR=Sphi(3); drsc; X23 = X;

% Yaw input
% ---------

nR=3;
aR=Spsi(1); drsc; X31 = X;
aR=Spsi(2); drsc; X32 = X;
aR=Spsi(3); drsc; X33 = X;

% Collective input
% ----------------

nR=4;
aR=Scol;    drsc; X41 = X;

% Wind Gust input
% ---------------
% Gust Response | States order:
%  1 :  4 - Swashplate actuators
%  5 : 15 - 6 DOF + rotor (outputs: theta, phi, psi)

Nsamp=round(tfg/dt+1);
nG=8; drsg; X51=X;
nG=7; drsg; X52=X;
nG=9; drsg; X53=X;

% ------------------------------------------------------------------------

% Spec 1
% ------

[pitch_dist,pitch_bw,pitch_pd] = d_bw_pd(A,B,C,D,1,1,PL1);
[roll_dist ,roll_bw ,roll_pd ] = d_bw_pd(A,B,C,D,2,2,PL1);
[yaw_dist  ,yaw_bw  ,yaw_pd  ] = d_bw_pd(A,B,C,D,3,3,PL1);

% Spec 2
% ------

[pitch_zeta,pitch_b] = zeta(A,B,C,D,1,1);
[roll_zeta ,roll_b ] = zeta(A,B,C,D,2,2);
[yaw_zeta  ,yaw_b  ] = zeta(A,B,C,D,3,4);

% Spec 3
% ------

pitch_r1 = max(X11(:,16))/Stet(1);
pitch_r2 = max(X12(:,16))/Stet(2);
pitch_r3 = max(X13(:,16))/Stet(3);
roll_r1  = max(X21(:,15))/Sphi(1);
roll_r2  = max(X22(:,15))/Sphi(2);
roll_r3  = max(X23(:,15))/Sphi(3);
yaw_r1   = max(X31(:,17))/Spsi(1);
yaw_r2   = max(X32(:,17))/Spsi(2);
yaw_r3   = max(X33(:,17))/Spsi(3);

% Spec 4
% ------

pitch_d = X23(:,19)/Sphi(3);
roll_d  = X13(:,18)/Stet(3);
yaw_d1 = X41(t3:nt,17);
 r1 = X41(t1,17);
 r3 = (yaw_d1(1)-r1)*sign(r1);
yaw_d2 = r3/abs(X41(t3,17));
yaw_d3 = r1/X41(t3,17);

% Spec 5
% ------

pitch_peak = X51(1:tg10-1,12);
pitch_ts   = X51(tg10:ntg,12);
roll_peak  = X52(1:tg10-1,11);
roll_ts    = X52(tg10:ntg,11);
yaw_peak   = X53(1:tg10-1,13);
yaw_ts     = X53(tg10:ntg,13);

% Saving Data
% -----------

save simu pitch_dist roll_dist yaw_dist ...
          pitch_bw   roll_bw   yaw_bw   ...
          pitch_pd   roll_pd   yaw_pd   ...
          pitch_b    roll_b    yaw_b    ...
          pitch_zeta roll_zeta yaw_zeta ...
          pitch_r1   roll_r1   yaw_r1   ...
          pitch_r2   roll_r2   yaw_r2   ...
          pitch_r3   roll_r3   yaw_r3   ...
          pitch_d    roll_d    yaw_d1 yaw_d2 yaw_d3 ...
          pitch_peak roll_peak yaw_peak ...
          pitch_ts   roll_ts   yaw_ts

diary off;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  end of simu.m  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [d,Bw,Pd] = d_bw_pd(A,B,C,D,Iu,Iy,PL);
%
% MATLAB function for ADOCS
% By : Gil Yudilevitch
% Last updated  :  11/19/92
%
% Calculate Band Width [rad/sec] and Phase Delay [sec] for
% the SISO system (input Iu output Iy) of:
%    dx/dt = Ax + Bu
%        y = Cx + Du
% Then calculate the distance of (Bw,Pd) to a given bound (spec)
%
%    [d,Bw,Pd] = d_bw_pd(A,B,C,D,Iu,Iy)
%

% Calculate Bw
% ------------

w=0; p=0;
for i = 1:5,
  epsilon = 0.1^(i-1);
  while p > -135,
    q=p;
    w = w+epsilon;
    p=angle(C(Iy,:)*inv(j*w*eye(size(A))-A)*B(:,Iu)+D(Iy,Iu))*180/pi;
    if p > 0, p=p-360; end;
  end;
  p=q;
  w = w-epsilon;
end;
Bw = w+epsilon/2;
```

```
% Calculate Pd
% ------------

p=angle(C(Iy,:)*inv((j*3*Bw*eye(size(A))-A))*B(:,Iu)+D(Iy,Iu))*180/pi;
if p > 0, p=p-360; end;
if p > -180,  Pd = 0;
else,
  p=0;
  for i = 1:5,
    epsilon = 0.1^(1-1);
    while p > -180,
      q=p;
      w = w+epsilon;
      p=angle(C(Iy,:)*inv((j*w*eye(size(A))-A))*B(:,Iu)+D(Iy,Iu))*180/pi;
      if p > 0, p=p-360; end;
    end;
    p=q;
    w = w-epsilon;
  end;
  w = w+epsilon/2;
  p=angle(C(Iy,:)*inv((j*2*w*eye(size(A))-A))*B(:,Iu)+D(Iy,Iu))*180/pi-360;

% Assumption, at 2*w180 : -180 deg  >=  phase >= -540 deg !!!

  Pd = -(180+p)*pi/360/w;
end;

% Calculate d
% -----------

d=Bw-polyval(PL,Pd);
if d > 0, d=0;
else,
  dmin=100;
  d=(d/4.45)^2;
  Pd0=Pd;
  while d < dmin,
    dmin=d;
    Pd0=Pd0-0.005;
    d=((Pd-Pd0)/0.4)^2+((Bw-polyval(PL,Pd0))/4.45)^2;
  end;
  d=dmin;
end;

%---------------------  End of d_bw_pd.m -------------------------


function [z,b]=zeta(A,B,C,D,Iu,Iy)
%
% MATLAB function for ADOCS
% By : Gil Yudilevitch
% Last updated  :  12/08/92
%
% Approximates the n-order SISO system (Iu input , Iy output)
% by second-order system and calculate the damping ratio.
% g(s) = k/(s^2 + a*S +b)
% z = a/(2*sqrt(abs(b)))
%
%    [Zeta,b] = zeta(A,B,C,D,Iu,Iy)
%

epsilon=1.0e-12;
Ts=0.25;                                          % Sampling_time
```



```
[Ad,Bd]=c2d(A,B(:,Iu),Ts); Cd=C(Iy,:); Dd=D(Iy,Iu); % Convert to dig (ZOH)
n=20;                                             % No. of ponts
u1=ones(n,1);
u=u1+0.1*randn(n,1);                              % Input = "noisey" step
[y,x]=dlsim(Ad,Bd,Cd,Dd,u);
th=arx([y,u],[2,2,1],4096,Ts);                    % System ID using ARX model
[Ni,Di]=contin(th);                               % Di(1)=1, Di(2)=a, Di(3)=b
z=Di(2)/(2*sqrt(epsilon+abs(Di(3))));             % Computing z
b=Di(3);

--------------------------  End of zeta.m  ----------------------------


function y=limit(l,u,x)
% Saturation function:
%
%            ^ y (output component)
%          u | /----                x,y,l,u are all vectors of the same length
%            |/
%      ------/------> x (input component)
%           /|
%      ____ / | l
%
% By: Gil Yudilevitch
% Updated: 09-22-92
% See also SAT.M (for scalar sequences)
%
%     y = limit(l,u,x)
%

n=length(l);
for i=1:n,
   if x(i) >= u(i), y(i)=u(i);
  else,
   if x(i) <= l(i), y(i)=l(i);
  else,         y(i)=x(i);
  end; end;
end;
y=y';

--------------------------  End of limit.m  ----------------------------


% ***************************************************************************
%   Digital Recursive Simulation Command (inputs)
%   Solve NON-LINEAR equations for simu.m (CONSOL-MATLAB file)
%   By : Gil Yudilevitch
%   Updated  :  02/04/93
% ***************************************************************************

Xf = zeros(Nsamp,7 );
Xs = zeros(Nsamp,4 );
Xp = zeros(Nsamp,11);
R=zeros(1,4); R(1,nR)=aR;

for k=1:Nsamp,

  Rm(k,:)  = Xf(k,:)*Cf' + R*Df';
  E(k,:)   = Rm(k,:) - Xp(k,:)*H';
  if Ndel(1) >= k, Ed(k,1) = E(1,1); else, Ed(k,1)=E(k-Ndel(1),1); end;
  if Ndel(2) >= k, Ed(k,2) = E(1,2); else, Ed(k,2)=E(k-Ndel(2),2); end;
  if Ndel(3) >= k, Ed(k,3) = E(1,3); else, Ed(k,3)=E(k-Ndel(3),3); end;
  if Ndel(4) >= k, Ed(k,4) = E(1,4); else, Ed(k,4)=E(k-Ndel(4),4); end;
  Rs(k,:)  = (limit(Csl,Csu,Ed(k,:)'))';
```

```
  Us(k,:)  = (limit(Rsl,Rsu,(Rs(k,:)-Xs(k,:))'))';

  Xf(k+1,:) = Xf(k,:)*Afd' + R*Bfd';
  Xs(k+1,:) = Xs(k,:)      + Us(k,:).*K*dt;
  Xp(k+1,:) = Xp(k,:)*Apd' + Xs(k,:)*Bpd';

end;

X=[Xf(1:Nsamp,:) Xs(1:Nsamp,:) Xp(1:Nsamp,:)];
```

```
/*=================================================*/
/*   ADOCS CONTROLLER - PDF File                   */
/*   By : Gil Yudilevitch                          */
/*   Last updated  :  09/09/92                     */
/*=================================================*/

design_parameter Kq
design_parameter Ktet
design_parameter Kp
design_parameter Kphi
design_parameter Kr
design_parameter Kpsi
design_parameter Mtet   init=   2.0
design_parameter Mphi   init=   2.5
design_parameter Mpsi   init=   2.0
design_parameter FGtet  init=  -0.329
design_parameter FGphi  init=   1.312
design_parameter FGpsi  init=   0.715
design_parameter FTtet  init=   0.519
design_parameter FTphi  init=   3.348
design_parameter FTpsi  init=   0.288

/* <<<<<<<<<<<     Initial Design Parameters    >>>>>>>>>>>>>>>>>>>  */

/* include "adocs.dp.open"   */ /* For open-loop parameters            */
/* include "adocs.dp.init"   */ /* For initial ADOCS design parameters */
   include "adocs.dp.final"     /* For final   ADOCS design parameters */
/* include "adocs.dp.1"      */ /* For continous run                   */

dt  = 1/30
tf  = 5
dtg = 1/30
tfg = 15

global double getout();

global double geto(name, t, dt)
global char *name;
global double t, dt;
global {
global    double r;
global    int i;
global    i = t/dt + 1.5;
global    r = getout(name, i);
global    return r;
global }

include "spec1.pit"
include "spec1.rol"
include "spec1.yaw"

include "spec2.pit"
include "spec2.rol"
include "spec2.yaw"

include "spec3.pit"
include "spec3.rol"
include "spec3.yaw"

include "spec4.pit"
include "spec4.rol"
include "spec4.yaw"

include "spec5.pit"
include "spec5.rol"
include "spec5.yaw"

/*=================  End of adocs.pdf  ============*/


/*=========================================================
/*  SPECIFICATION 1 - Small Amplitude, Short Term Response
/*  Bandwidth & Phasedelay
/*  Pitch (ref. ADS-33C  3.3.2.1)
/*  Included file of ADOCS.PDF
/*  By : Gil Yudilevitch
/*  Last update  :  09/09/92
/*=========================================================

constraint  "pit bw pd" soft
      {
      return getout("pitch_dist", 1);
      }
   <=
   good_value = 0.000
   bad_value  = 0.002

/*=================    END OF spec1.pit   ================


/*=========================================================
/*  SPECIFICATION 2 - Small Amplitude, Mid Term Response
/*  Damping Ratio
/*  Pitch (ref. ADS-33C  3.3.2.2)
/*  Included file of ADOCS.PDF
/*  By : Gil Yudilevitch
/*  Last update :  11/11/92
/*=========================================================

constraint "pit damp" soft
      {
      return getout("pitch_zeta", 1);
      }
   >=
   good_value = 0.35
   bad_value =  0.30

constraint "pit stab" hard
      {
      return getout("pitch_b", 1);
      }
   >=
   good_value = 0.001
   bad_value =  0.0009

/*==================    END OF spec2.pit   ====================*/


/*=============================================================*/
/*  SPECIFICATION 3 - Moderate Amplitude, Attitude Quickness   */
/*  q_pk/theta_pk                                              */
/*  Pitch (ref. ADS-33C  3.3.3)                                */
/*  Included file of ADOCS                                     */
/*  By : Gil Yudilevitch                                       */
/*  Last update :  03/02/93                                    */
/*=============================================================*/

constraint  "pit quick1" soft
```

```
      (
      return getout("pitch_r1", 1);
      )
   >=
   good_value = 0.70
   bad_value  = 0.63

constraint  "pit quick2" soft
      (
      return getout("pitch_r2", 1);
      )
   >=
   good_value = 0.40
   bad_value  = 0.36

constraint  "pit quick3" soft
      (
      return getout("pitch_r3", 1);
      )
   >=
   good_value = 0.25
   bad_value  = 0.22

/*==================    END OF spec3.pit   ====================*/


/*============================================================*/
/*  SPECIFICATION 4 - Decoupling                              */
/*  theta/delta_phi                                           */
/*  Pitch (ref. ADS-33C  3.3.9.2)                             */
/*  Included file of ADOCS.PDF                                */
/*  By : Gil Yudilevitch                                      */
/*  Last update  :  12/23/92                                  */
/*============================================================*/

functional_constraint "pit dec up" soft
   for t from 0 to tf by dt
   (  import dt;
      return geto("pitch_d", t, dt); )
 <= good_curve = ( return  0.25; )
    bad_curve =  ( return  0.30; )

functional_constraint "pit dec lo" soft
   for t from 0 to tf by dt
   (  import dt;
      return geto("pitch_d", t, dt); )
 >= good_curve = ( return -0.25; )
    bad_curve =  ( return -0.30; )

/*==================    END OF spec4.pit   ====================*/


/*============================================================*/
/*  SPECIFICATION 5 - Wind-Gust Rejection                     */
/*  theta/gust                                                */
/*  Pitch (ref. M. Tischler 9/26/90)                          */
/*  Included file of ADOCS.PDF                                */
/*  By : Gil Yudilevitch                                      */
/*  Last update  :  02/23/92                                  */
/*============================================================*/

functional_constraint "p gust p u" soft
   for t from 0 to 10-dtg by dtg
   (  import dtg;
      return geto("pitch_peak", t, dtg); )
 <= good_curve = ( return  0.065; )
    bad_curve =  ( return  0.075; )

functional_constraint "p gust p l" soft
   for t from 0 to 10-dtg by dtg
   (  import dtg;
      return geto("pitch_peak", t, dtg); )
 >= good_curve = ( return -0.065; )
    bad_curve =  ( return -0.075; )

functional_constraint "p gust t u" soft
   for t from 10 to tfg by dtg
   (  import dtg;
      return geto("pitch_ts", t, dtg); )
 <= good_curve = ( return  0.0065; )
    bad_curve =  ( return  0.0075; )

functional_constraint "p gust t l" soft
   for t from 0 to tfg by dtg
   (  import dtg;
      return geto("pitch_ts", t, dtg); )
 >= good_curve = ( return -0.0065; )
    bad_curve =  ( return -0.0075; )

/*==================    END OF spec5.pit   ====================*/
```

Appendix C -
Stabilization via Smooth Optimization: Continuous-Time Case

Chin-Yee Lin Michael K.H. Fan

School of Electrical Engineering
Georgia Institute of Technology, Atlanta, GA 30332

Abstract

Let $A(x)$ be an $n \times n$ real matrix whose elements are analytic functions of $x \in \mathbf{R}^m$. In this paper we consider the problem on minimizing the largest real part of the eigenvalues of $A(x)$. This problem is in general non-differentiable and non-convex. It is shown that an existing algorithm proposed by Fan [2] for solving a class of smooth constrained problems can be applied here. This algorithm enjoys the property that, if started close enough to a local minimizer x^*, then it will converge to x^* quadratically. To ensure numerical stability and efficiency, our derivations employ real Schur decomposition instead of eigenvalue decomposition of $A(x)$. We also develop result on the analyticity and uniqueness of the real Schur decomposition of $A(x)$. This result may be of interest on its own right.

C.0 Notation

I	Identity matrix of appropriate size
x^T	Transpose of vector x
$\|x\|$	Euclidean norm of vector x
A^T	Transpose of matrix A
$A > 0$	Matrix A is positive definite ($A \geq 0$, $A < 0$, and $A \leq 0$ are defined similarly)
$\mathrm{tr}(A)$	Trace of matrix A
$A^\dagger$	Moore-Penrose generalized inverse of matrix A
$\Re$	real part

C.1 Introduction

Let $A(x)$ be an $n \times n$ real matrix whose elements are analytic functions of $x \in \mathbf{R}^m$. The eigenvalues of $A(x)$ are denoted by $\lambda_1(x), \ldots, \lambda_n(x)$, arranged in decreasing order by their real parts, i.e.,

$$\Re\lambda_1(x) \geq \cdots \geq \Re\lambda_n(x)$$

(those eigenvalues with same real parts but different imaginary parts may be ordered arbitrarily). Define

$$\phi(x) = \max_{i=1,\ldots,n} \Re\lambda_i(x)$$

So $\phi(x)$ is the largest real part of eigenvalues of $A(x)$. In this paper, we study the optimization problem

$$\phi^* := \min_{x \in \mathbf{R}^m} \phi(x) \tag{1.1}$$

which arises in stabilization of dynamic systems. For instance, consider the nonlinear system

$$\dot{z}(t) = A(x)z(t) + f(x, z(t)), \quad z(0) = z_0 \tag{1.2}$$

where $f(x, z(t))$ denotes the second or higher order terms of the state $z(t)$, and x is the decision variable to be chosen in order to meet some design specifications. The system (1.2) is said to be asymptotically stable at the origin if there exists $\delta > 0$ such that $\|z_0\| < \delta$ implies $\|z(t)\| \to 0$ as $t \to 0$. It is well-known that (1.2) is asymptotically stable at the origin if $A(x)$ is a stable matrix, i.e., $\phi(x) < 0$. Thus, if $\phi^* < 0$, then we could choose some x to guarantee the stability of (1.2).

The function $\phi(x)$ is in general non-differentiable. Typically, the process of minimization tends to make at least the real parts of eigenvalues coalesce at the solution. Moreover, near the solution when it is non-differentiable, it may not be possible to represent $\phi(x)$ as the maximum of finitely many differentiable functions; this type of non-differentiability usually makes the problem difficult to solve.

The function $\phi(x)$ in general has local minimizers which are not global. Therefore, obtaining a global minimizer will typically require massive computation. If $A(x)$ is symmetric and depends

upon x affinely, then $\phi(x)$ is convex in x. In this case, among many other algorithms, a quadratically convergent local algorithm has been proposed in [2] for solving (1.1).

In this paper, we shall only be concerned with finding a local solution of (1.1). It is shown that the algorithm in [2] can be extended here such that, if started close enough to a local minimizer x^*, then the proposed algorithm will converge to x^* quadratically. The extension involves two key steps. First, we show that it is possible to formulate (1.1) near a local solution as a smooth constrained problem which satisfies the hypotheses given in [2]. For this purpose, we use some results in analytic perturbation of eigenvalues, e.g., the Puiseux series representation of repeated eigenvalues. Second, we show how to derive first and second derivatives for some functions (which involve eigenvalues of $A(x)$) to be used in the algorithm. To ensure numerical stability and efficiency, our derivation is based on real Schur decomposition of $A(x)$ rather than its eigenvalue decomposition. We also developed result on the analyticity and uniqueness of real Schur decomposition of $A(x)$. This result may be of interest on its own right.

The paper is organized as follows. In Section C.2, we give the assumptions that will be used throughout the paper. In Section C.3, we show the smooth formulation of (1.1). In Section C.4, we briefly describe the algorithm proposed in [2] and discuss how it can be extended to solve (1.1). Section C.5 is devoted to the derivatives of real Schur decomposition of $A(x)$ as well as the functions used in the proposed algorithm. Finally, numerical examples are given in Section C.6 to demonstrate the proposed algorithm.

C.2 Assumptions

In this section, we give the assumptions to be used throughout the paper.

- $A(x)$ is analytic for all $x \in \mathbb{R}^m$.
- There exists a local solution x^* of (1.1).
- Suppose that

  $$\Re\lambda_1(x^*) = \ldots = \Re\lambda_q(x^*) > \Re\lambda_{q+1}(x^*)$$

  for some q. Then, the subset of eigenvalues $\{\lambda_1(x^*), \ldots, \lambda_q(x^*)\}$ is non-defective, i.e., there are q linearly independent eigenvectors associated with those eigenvalues. Furthermore, let Z_i denote the component matrix (see, e.g., [5] for the definition of the component matrix) associated with $\lambda_i(x^*)$. Then, for $i = 1, \ldots, q$ and $j = 1, \ldots, m$, the set of eigenvalues of the matrix

  $$Z_i \frac{\partial A(x^*)}{\partial x_j} Z_i$$

  is non-defective.

C.3 A smooth formulation

Let x^* denote a local solution of (1.1). Given $x \in \mathbb{R}^m$, the multiplicity of $\phi(x)$ is said to be q if $\Re\lambda_1(x) = \Re\lambda_q(x) > \Re\lambda_{q+1}(x)$. Suppose that the multiplicity q^* of $\phi(x^*)$ has been correctly identified. In this section, we give a smooth formulation of (1.1) near x^*. Specifically, we define certain functions $f_1(x)$ and $f_2(x)$, both map from $\mathbb{R}^m$ to $\mathbb{R}$, that satisfy the following properties: (i) x^* is a local solution the optimization problem

$$\min_{x\in\mathbb{R}} \{f_1(x) : f_2(x) = 0\} \tag{3.1}$$

(ii) $f_1(x)$ and $f_2(x)$ are twice continuously differentible at x^*, and (iii) $f_1(x)$ and $f_2(x)$ satisfy the assumptions required by the algorithm in [2] ((iii) will be checked after a recall of the algorithm in [2]). In next section, we show how to solve (1.1) by solving (3.1) using the algorithm in [2].

Let us first proceed with an example to illustrate the main ideas in defining such $f_1(x)$ and $f_2(x)$. Suppose that q^* is 5, i.e.,

$$\Re\lambda_1(x^*) = \cdots = \Re\lambda_5(x^*) > \Re\lambda_6(x^*)$$

Assume that the complex conjugate pairs $\{\lambda_1(x^*), \lambda_2(x^*)\}$ and $\{\lambda_1(x^*), \lambda_2(x^*)\}$ are nonreal and identical, and $\lambda_5(x^*)$ is real. Since q^* is known by assumption, x^* will still be a local solution of (1.1) even if the following constraint is imposed

$$\Re\lambda_1(x) = \cdots = \Re\lambda_5(x) \tag{3.2}$$

Also, it is easy to check that, if restricted in the constraint set defined by (3.2), then x^* is a local minimizer of $\phi(x)$ if and only if it is a local minimizer of $\sum_{i=1}^5 \Re\lambda_i(x)$. Therefore, if we define

$$f_1(x) = \frac{1}{5}\sum_{i=1}^{5} \Re\lambda_i(x) = \frac{1}{5}\sum_{i=1}^{5} \lambda_i(x) \tag{3.3}$$

and

$$f_2(x) = \sum_{i=1}^{4} (\Re\lambda_i(x) - \Re\lambda_{i+1}(x))^2 \tag{3.4}$$

Then, we have the assertion that x^* is also a local solution of (3.1).

Now let us turn to the question on differentiability of $f_1(x)$ and $f_2(x)$. The function $f_1(x)$ defined in (3.3) is analytic at x^* (see below). However, the function $f_2(x)$ defined in (3.4) is not even differentiable. Roughly speaking, this is because that the expression in (3.4) is not "symmetric" with respect to $\Re\lambda_1$ and $\Re\lambda_3$ or with respect to $\Re\lambda_2$ and $\Re\lambda_4$ (since $\lambda_1(x^*) = \lambda_3(x^*)$ and $\lambda_2(x^*) = \lambda_4(x^*)$ by assumption). To make it "symmetric", we may add a few redundant terms, i.e., re-define $f_2(x)$ by

$$f_2(x) = \sum_{i=1}^{4}\sum_{j=i+1}^{5} (\Re\lambda_i(x) - \Re\lambda_j(x))^2 \tag{3.5}$$

It is easy to see that (3.4) is zero if and only if (3.5) is. However, the difference it makes is that $f_2(x)$ defined by (3.5) is twice continuously differentiable at x^*. This will be shown below.

Of course, there are many other ways to define analytic $f_1(x)$ or $f_2(x)$, besides simply multiplying them by constants. For example, we may define $f_2(x)$ to be the square of the right hand side of (3.5). Or, on the right hand side of (3.5), we may use other positive even powers other than squares. However, among all $f_1(x)$ and $f_2(x)$ that satisfy the required properties, it seems that (3.3) together with (3.5) is the most convenient pair as it requires the least information for the eigenvalue locations. This point shall become clear later. In fact, so far, only (3.3) together (3.5) satisfies all the require properties.

We are now ready to give the main result of this section.

Theorem 3.1. Let x^* denote a local solution of (1.1). Suppose that the multiplicity q^* of $\phi(x^*)$ has been correctly identified. Define

$$f_1(x) = \frac{1}{q^*} \sum_{i=1}^{q^*} \lambda_i(x) \tag{3.6}$$

and

$$f_2(x) = \sum_{i=1}^{q^*-1} \sum_{j=i+1}^{q^*} (\Re\lambda_i(x) - \Re\lambda_j(x))^2 \tag{3.7}$$

Then, x^* is a local solution of (3.1). Also, at x^*, $f_1(x)$ is analytic and $f_2(x)$ is twice continuously differentiable. □

C.4 Fan's algorithm

In this section, we give a brief description of Fan's algorithm. It will be used in solving (2.1). More details about the algorithm can be found in [2].

Let x^* be a local solution of (1.1). Suppose that the multiplicity q^* of $\phi(x^*)$ has been correctly identified. Recall that the functions $f_1(x)$ and $f_2(x)$ defined in Section C.2 are twice continuously differentiable at x^*. Therefore, if both $x - x^*$ and h are sufficiently small, then $f_i(x+h)$, $i = 1, 2$, is equal to its Taylor series expansion about x, i.e.,

$$f_i(x+h) = f_i(x) + g_i^T(x)h + \frac{1}{2}h^T H_i(x)h + O(\|h\|^3), \qquad i = 1, 2$$

where $g_i(x)$ and $H_i(x)$ are the gradient and Hessian of $f_i(x)$, respectively. Expressions of $g_i(x)$ and $H_i(x)$, $i = 1, 2$, will be given in Section C.5, where the following properties they possess can be easily verified. (i) $g_2(x) = 0$ if $f_2(x) = 0$. (ii) $H_2(x) = H_{21}(x) + H_{22}(x)$; $H_{21}(x)$ and $H_{22}(x)$ are symmetric; $H_{21}(x) \geq 0$; $H_{22}(x) = 0$ if $f_2(x) = 0$.

Let the columns of $N(x)$ (resp. $R(x)$) form an orthonormal basis for the null (resp. range) space of $H_{21}(x)$. Since $H_{21}(x)$ is symmetric, we have the identity

$$N(x)N^T(x) + R(x)R^T(x) = I$$

for every x in a neighborhood of x^*.

It is shown that under certain regularity conditions, a local solution x^* satisfies

$$\begin{cases} N^T(x^*)g_1(x^*) = 0 \\ R^T(x^*)g_2(x^*) = 0 \end{cases} \tag{4.1}$$

which is a system of m nonlinear equations with m unknowns [2, 8]. One step of Fan's algorithm consists in linearization of the nonlinear equations (4.1) at the current point x^k and chooses the new point x^{k+1} as the solution of the obtained system of linear equations. It can be shown that the matrices $N(x)$ and $R(x)$ can be chosen as smooth functions of x. It follows by standard arguments that if started closed enough to a local minimizer x^*, then it will converge to x^* quadratically.

Given $h \in \mathbb{R}^m$, define $D(x,h)$ to be the directional derivative of $H_{21}(x)$ in the direction h. Also, let $\hat{D}(\cdot,\cdot) : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^{m\times m}$ be such that $D(x,h)\phi = \hat{D}(x,\phi)h$ for all $x,h,\phi \in \mathbb{R}^m$ (this is possible since $D(x,h)\phi$ is bilinear in h and ϕ). We now summarize Fan's algorithm as follows.

Algorithm 4.1. Let $\phi_2(x) = -H_{21}^{\dagger}(x)g_2(x)$. Define $h(x) = \phi_2(x)$ if $N(x)$ is a null matrix. Otherwise, let $\phi_1(x) = -H_{21}^{\dagger}(x)g_1(x)$, $Q(x) = H_1(x) + \hat{D}(x,\phi_1(x))$, and define

$$h(x) = \phi_2(x) - N(x)\left(N^T(x)Q(x)N(x)\right)^{-1} N^T(x)(Q(x)\phi_2(x) + g_1(x))$$

Then, x_{new} is defined by $x_{\text{new}} = x + h(x)$. □

C.5 Derivatives of eigenvalues

Since $A(x)$ is analytic, it is well-known that if $A(a)$ has all distinct eigenvalues, then the eigenvalues of $A(x)$, as functions of x, may be ordered in such a way that they are analytic at $x = a$. In this section, we assume that $A(a)$ has all distinct eigenvalues and derive its derivatives. The result will be used in the next section for the derivatives of $f_1(x)$ and $f_2(x)$ (which are required in the proposed algorithm for solving (2.1)).

To obtain the derivatives, we may proceed with an eigenvalue decomposition of $A(a)$, i.e., to have a nonsingular matrix T such that the similarity transformation $T^{-1}A(a)T$ is diagonal. The advantages of being able to use a diagonal matrix instead of a full one are obvious. However, we choose not to do so. This is because that the matrix T may be ill-conditioned in the sense that its inverse may be large. In this case, it is well-known that the similarity transformation will magnify significantly any error occurred in the matrix $A(a)$ and thus may cause numerical

instability. Moreover, we have to carry out the computation in complex arithmetics even though $A(a)$ is real.

Instead, we use a real Schur decomposition of $A(x)$, i.e., to have an orthogonal matrix U such that the orthogonal similarity transformation $S = U^T A(a)U$ is in a certain block upper triangular form, where the diagonal elements contain information about the eigenvalues (see below). The matrix S will be called a real Schur form of $A(a)$. Orthogonal similarity transformation is particularly desirable since neither U nor its inverse U^T can be large. In fact, it is easy to check that no element of U or its inverse can be greater than one in absolute value. Also, real arithmetics can be used throughout the computation.

However, nothing comes for free. Unlike in the case of eigenvalue decomposition, even with the order of eigenvalues specified, the matrix S in real Schur decomposition is non-unique (in fact, highly non-unique). In order to have the derivatives of eigenvalues using the matrices from a real Schur decomposition, we need first address the following questions. (i) Given a real Schur decomposition of $A(a)$

$$A(a) = U_a S_a U_a^T \tag{5.1}$$

is there a choice of real Schur decompositions of $A(x)$

$$A(x) = U(x)S(x)U(x)^T \tag{5.2}$$

such that (5.2) coincides with (5.1) at $x = a$, and both $U(x)$ and $S(x)$ are analytic at a ? (ii) If the answer to (i) is yes, then is the choice unique ? We will show that the answer to (i) is indeed affirmative. Also, with a mild assumption, the answer to (ii) is also affirmative. In fact, with the same assumption, the matrix $S(x)$ in (5.2) will be shown to have identical strictly lower triangular part for all x in some neighborhood of a. This property enables us to compute the derivatives of the eigenvalues using real Schur decompositions.

We continue with a review of real Schur decomposition. Although a proof of real Schur decomposition can be easily found in many books in matrix computation, we include one here (which is borrowed from [3]) since the proof itself is useful in our subsequent developments.

C.5.1 Real Schur decomposition

Fact 5.1. (see, e.g., [3]) Let $A \in \mathbf{R}^{n\times n}$. Then there is a real orthogonal matrix $U \in \mathbf{R}^{n\times n}$ such that $U^T AU = S$ is block triangular with 1×1 and 2×2 blocks on its diagonal. The 1×1 blocks contain the real eigenvalues of A, and the eigenvalues of the 2×2 blocks are the complex eigenvalues of A. Furthermore, the diagonal blocks may be arranged in any prescribed order.

Proof. It is algorithmic and proceeds by a sequence of reduction of similar type. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A, arranged in any prescribed order (with complex pairs ordered together).

Suppose first that λ_1 is a real with a normalized eigenvector v_1. Choose $z_1, \ldots, z_{n-1} \in \mathbb{R}^n$ such that $\{v_1, z_1, \ldots, z_{n-1}\}$ form a basis of $\mathbb{R}^n$. Apply the Gram-Schmidt orthonormalization procedure to this basis to produce an orthonormal basis $\{v_1, w_1, \ldots, w_{n-1}\}$ of $\mathbb{R}^n$. Define $U_1 = [v_1\ w_1\ \ldots\ w_{n-1}]$. Then, $U_1^T A U_1$ has the form

$$U_1^T A U_1 = \left[\begin{array}{c|c} \Lambda_1 & * \\ \hline 0 & A_1 \end{array}\right] \tag{5.3}$$

where $\Lambda_1 = \lambda_1$ and $A_1 = [w_1 \cdots w_{n-1}]^T A [w_1 \cdots w_{n-1}] \in \mathbb{R}^{(n-1)\times(n-1)}$ with eigenvalues $\lambda_2, \ldots, \lambda_n$. If $\lambda_1 = \alpha + i\beta$ is nonreal with nonzero eigenvector $v_1 = \mu + i\nu$. It can be shown that μ and ν are linearly independent. Choose $z_1, \ldots, z_{n-2} \in \mathbb{R}^n$ such that $\{\mu, \nu, z_1, \ldots, z_{n-2}\}$ form a basis of $\mathbb{R}^n$. Apply the Gram-Schmidt orthonormalization procedure to this basis to produce an orthonormal basis $\{\hat{\mu}, \hat{\nu}, w_1, \ldots, w_{n-2}\}$ of $\mathbb{R}^n$. Define $U_1 = [\hat{\mu}\ \hat{\nu}\ w_1\ \ldots\ w_{n-2}]$. Then, $U_1^T A U_1$ also has the form (5.3) where this time Λ_1 is a 2×2 real matrix whose eigenvalues are λ_1 and λ_2 ($\lambda_2 = \overline{\lambda}_1$), and $A_1 = [w_1 \cdots w_{n-2}]^T A [w_1 \cdots w_{n-2}] \in \mathbb{R}^{(n-2)\times(n-2)}$ with eigenvalues $\lambda_3, \ldots, \lambda_n$. Now do it over again for A_1 and determine an orthogonal matrix $U_2 \in \mathbb{R}^{(n-1)\times(n-1)}$ such that

$$U_2^T A_1 U_2 = \left[\begin{array}{c|c} \Lambda_2 & * \\ \hline 0 & A_2 \end{array}\right] \tag{5.4}$$

Define

$$V_2 = \left[\begin{array}{c|c} I & 0 \\ \hline 0 & U_2 \end{array}\right]$$

Then the matrices V_2 and $U_1 V_2$ are orthogonal, and $(U_1 V_2)^T A (U_1 V_2)$ has the form

$$(U_1 V_2)^T A (U_1 V_2) = \left[\begin{array}{cc|c} \Lambda_1 & * & * \\ 0 & \Lambda_2 & \\ \hline & 0 & A_2 \end{array}\right] \tag{5.5}$$

Continue this reduction to produce orthogonal matrices $U_i \in \mathbb{R}^{(n-i)\times(n-i)}$ and $V_i \in \mathbb{R}^{n\times n}$, $i = 1, \ldots, s$, where s denotes the number of eigenvalues of A with non-negative imaginary parts. Then, the matrix $U = U_1 V_2 V_3 \cdots V_s$ is orthogonal and $U^T A U$ yields the desired form. □

It is noted that the orthogonal matrix U and the block triangular matrix S in the real Schur decomposition are by far non-unique; we have not only the freedom in choosing the order of the eigenvalues, but also, in each reduction step, the freedom in choosing an eigenvector and vectors z_i's.

In the sequel, we will denote by $\mathcal{S}_n$ the set of $n \times n$ matrices which are real Schur forms of themselves, and satisfy the following properties: (i) the diagonal blocks of any element in $\mathcal{S}_n$ are arranged in descending order according to the real parts of its eigenvalues (those blocks with same

real parts of eigenvalues may be ordered arbitrarily), and (*ii*) every 2×2 diagonal block, if any, of $S \in \mathcal{S}_n$ has different diagonal elements, i.e., if

$$\begin{bmatrix} s_1 & s_2 \\ s_3 & s_4 \end{bmatrix}$$

is a diagonal block of S, then $s_1 \neq s_4$.

By the virtue of Fact 5.1, every matrix has a real Schur form which satisfies the property (*i*). This property will be shown to greatly simplify the expression for the derivatives of $f_1(x)$. On the other hand, not every matrix has a real Schur form which satisfies the property (*ii*). For example, let

$$A = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \tag{5.6}$$

Then, A has at most two real Schur forms (itself and its transpose). In fact, it can be easily checked that every 2×2 real matrix is of the form (5.6) if and only if it is a normal matrix (a real matrix A is normal if $AA^T = A^TA$). Furthermore, if none of 2×2 diagonal blocks of a real Schur form of A is normal, then it can be shown that A has a real Schur form that satisfies the property (*ii*).

C.5.2 Derivatives of real Schur form

We first address the analyticity of the real Schur form.

Theorem 5.1. Suppose that $A(a)$ has all distinct eigenvalues. Then the following statements hold. (*i*) There exist analytic functions $U(x)$ and $S(x)$, such that $U(x)S(x)U^T(x)$ constitutes a real Schur decomposition of $A(x)$ in an open neighborhood of a. (*ii*) Let $U_aS_aU_a^T$ be a real Schur decomposition of $A(a)$ and assume $S_a \in \mathcal{S}_n$. Then the functions $U(x)$ and $S(x)$ in (*i*) can be chosen in such a way that $U(a) = U_a$, $S(a) = S_a$, and the strictly lower triangular part of $S(x)$ is identical in an open neighborhood of a. (*iii*) The choice for $U(x)$ and $S(x)$ in (*ii*) is unique.

Proof. Let U_1 be given as in the proof of Fact 5.1 for the real Schur decomposition of $A(a)$. Also let v_1 and $z_1, \ldots, z_{n-1}$ be the corresponding choice in the first step of the reduction procedure. We show first that, for all x in some open neighborhood of a, there exists an analytic function $U_1(x)$ such that $U_1(a) = U_1$ and the matrix $U_1^T(x)A(x)U_1(x)$ has the form (5.3) or (5.4). Since all eigenvalues of $A(a)$ are distinct, it is well-known that the eigenvector $v_1(x)$, with $v_1(a) = v_1$, can be chosen to be analytic at a [4]. Now, fix $z_1, \ldots, z_{n-1}$ and consider the matrix $U_1(x)$ in a neighborhood of a. It is easily checked that, in a neighborhood of a, the vectors $v_1(x), z_1, \ldots, z_{n-1}$ still form a basis for $\mathbf{R}^n$. Also, the Gram-Schmidt orthonormalization procedure, as a function of v_1, is analytic at $v_1(a)$. Consequently, $U_1(x)$ is also analytic at a (since it is a composition of two analytic functions). Following similar arguments, it can be shown that $U(x)$ and $S(x) = U^T(x)A(x)U(x)$ may be chosen to have the desired property in the first claim.

To show the second claim, without loss of generality, we may assume that $A(x)$ is a 2×2 matrix with nonreal eigenvalues (so it is already in a real Schur form). Let

$$A(x) = \begin{bmatrix} a_1(x) & a_2(x) \\ a_3(x) & a_4(x) \end{bmatrix}$$

In view of the assumption, we have $a_1(a) \neq a_4(a)$. Let

$$t(x) = \sqrt{(a_1(x) - a_4(x))^2 + (a_2(x) + a_3(x))^2}$$

Then, it is tedious but straightforward to check that

$$\theta_x = \frac{1}{2}\left(\sin^{-1}\frac{2a_3(a) - a_3(x) + a_2(x)}{t(x)} - \cos^{-1}\frac{a_4(x) - a_1(x)}{t(x)}\right)$$

and the orthogonal matrix

$$U(x) = \begin{bmatrix} \cos\theta_x & -\sin\theta_x \\ \sin\theta_x & \cos\theta_x \end{bmatrix}$$

are all analytic at a, and

$$U^T(x)A(x)U(x) = S(x) = \begin{bmatrix} * & * \\ a_3(a) & * \end{bmatrix}$$

The proof for the third claim will given at the end of this section. □

Lemma 5.1. Let $S \in \mathcal{S}_n$ and $B \in \mathbb{R}^{n\times n}$. Assume that S has all distinct eigenvalues. Then, among all skew-symmetric matrices in $\mathbb{R}^{n\times n}$, there exists a unique P such that $P^TS + SP + B$ is upper triangular.

Proof. We show the claim by a recursive construction. Let $e_1, \ldots, e_n$ denote the standard basis vectors in $\mathbb{R}^n$. For $i = 1, \ldots, n-1$, define $E_i = [e_{i+1} \cdots e_n] \in \mathbb{R}^{n\times(n-i)}$, $p_i = E_i^TPe_i \in \mathbb{R}^{n-i}$, and $S_i = E_i^TSE_i \in \mathbb{R}^{(n-i)\times(n-i)}$. Thus, any skew-symmetric matrix P is uniquely determined by the vectors $p_1, \ldots, p_{n-1}$. Suppose that $p_1, \ldots, p_{k-1}$ have been obtained for some k. We then show how to proceed. Let $S = \{s_{ij}\}$. First, let us assume $s_{k,k-1} = s_{k+1,k} = 0$ (so, s_{kk} is an eigenvalue of S). Since $P^TS + SP + B$ is upper triangular, we have

$$E_k^T\left(P^TS + SP + B\right)e_k = 0$$

which is equivalent to

$$-\sum_{i=1}^{k-1} s_{ik}E_k^TP^Te_i - (s_{kk}I - S_k)p_k + E_k^TBe_k = 0$$

Thus, p_k can be solved and is equal to

$$p_k = (s_{kk}I - S_k)^{-1}\left(-\sum_{i=1}^{k-1} s_{ik}E_k^TP^Te_i + E_k^TBe_k\right) \tag{5.7}$$

Notice that the right hand side of (5.8) does not depend upon $p_k, \ldots, p_{n-1}$. Also, since S has distinct eigenvalues by assumption, the matrix $s_{kk}I - S_k$ in (5.7) is invertible.

Second, we assume $s_{k-1,k} = 0$ and $s_{k+1,k} \neq 0$. Again, since $P^TS + SP + B$ is upper triangular, we have

$$\begin{cases} E_k^T \left(P^TS + SP + B\right) e_k & = 0 \\ E_{k+1}^T \left(P^TS + SP + B\right) e_{k+1} & = 0 \end{cases}$$

which is equivalent to

$$\begin{cases} -\sum_{i=1}^{k-1} s_{ik} E_k^T P^T e_i - (s_{kk}I - S_k)p_k - s_{k+1,k} E_k^T P^T e_{k+1} + E_k^T B e_k & = 0 \\ -\sum_{i=1}^{k-1} s_{i,k+1} E_{k+1}^T P^T e_i - (s_{k+1,k+1}I - S_{k+1})p_{k+1} - s_{k,k+1} E_{k+1}^T P^T e_k + E_{k+1}^T B e_{k+1} & = 0 \end{cases}$$

or

$$\begin{bmatrix} p_k \\ p_{k+1} \end{bmatrix} = \Phi^{-1} \begin{bmatrix} -\sum_{i=1}^{k-1} s_{ik} E_k^T P^T e_i + E_k^T B e_k \\ -\sum_{i=1}^{k-1} s_{i,k+1} E_{k+1}^T P^T e_i + E_{k+1}^T B e_{k+1} \end{bmatrix} \tag{5.8}$$

where

$$\Phi = \left[\begin{array}{c|ccccccc} c_1 & c_2 & \cdots & c_n & 0 & \cdots & 0 \\ \hline 0 & & & & & & \\ \vdots & & & \begin{bmatrix} s_{kk} & s_{k+1,k} \\ s_{k,k+1} & s_{k+1,k+1} \end{bmatrix} \oplus (-S_{k+1}) & & & \\ 0 & & & & & & \end{array}\right]$$

$c_i = s_{kk} - s_{k+1,k+i}$, and $\oplus$ denotes the Kronecker sum. Again, it is easy to check that the right hand side of (5.8) does not depend upon $p_{k+2}, \ldots, p_n$. Also, in view of the definition of $\mathcal{S}_n$ (which implies $c_1 = s_{kk} - s_{k+1,k+1} \neq 0$), the assumption that S has distinct eigenvalues, and the eigenvalues of Kronecker sum of two matrices, we conclude that Φ is invertible. Finally, the uniqueness of P is obvious from the construction. □

In the sequel, given $S \in \mathcal{S}_n$ with distinct eigenvalues and $B \in \mathbb{R}^{n\times n}$, we will denote by

$$P = \mathcal{A}(S, B) \tag{5.9}$$

the unique skew-symmetric matrix P that constructed by the recursive procedure given in the proof of Lemma 5.1. Using (5.9), we show below how to derive the first and second order derivatives of the real Schur form.

Theorem 5.2. Suppose that $A(x)$ is analytic at $a \in \mathbb{R}^m$ and $A(a)$ has all distinct eigenvalues. Let $U(a)S(a)U^T(a)$ be a real Schur decomposition of $A(a)$ and assume $S(a) \in \mathcal{S}_n$. Let $U(x)$ and $S(x)$ be some analytic functions given in Theorem 5.1 (*ii*). For $i, j = 1, \ldots, m$, define $B_i = U^T(a)\frac{\partial A(a)}{\partial x_i}U(a)$ and $P_i = \mathcal{A}(S(a), B_i)$. Then, for $i = 1, \ldots, m$, it holds that

$$\frac{\partial S(a)}{\partial x_i} = P_i^T S(a) + S(a)P_i + B_i$$

Furthermore, for $i, j = 1, \ldots, m$, define

$$B_{ij} = P_j^T \frac{\partial S(a)}{\partial x_i} + \frac{\partial S(a)}{\partial x_i} P_j + P_i^T B_j + B_j P_i + U^T(a) \frac{\partial^2 A(a)}{\partial x_i \partial x_j} U(a)$$

and $P_{ij} = \mathcal{A}(S(a), B_{ij})$. Then, for $i, j = 1, \ldots, m$, it holds that

$$\frac{\partial^2 S(a)}{\partial x_i \partial x_j} = P_{ij}^T S(a) + S(a) P_{ij} + B_{ij}$$

Proof. For $i = 1, ..., m$, let $\frac{\partial U(x)}{\partial x_i} = U(x) P_i(x)$ for some $P_i(x) \in \mathbb{R}^{n \times n}$. It is clear that $P_i(x)$ is also analytic. Differentiating the identity $U^T(x)U(x) = I$ with respect to x_i yields

$$P_i^T(x) + P_i(x) = 0$$

i.e., $P_i(x)$ is skew-symmetric for all x. Also, differentiating the identity $S(x) = U^T(x)A(x)U(x)$ with respect to x_i at a yields

$$\frac{\partial S(a)}{\partial x_i} = P_i^T(a) S(a) + S(a) P_i(a) + B_i$$

By the property (ii) concerning $S(x)$ in Theorem 5.1, we have that $\frac{\partial S(a)}{\partial x_i}$ is an upper triangular matrix. In view of Lemma 5.1, $P_i(a)$ is unique and equal to $P_i(a) = \mathcal{A}(S(a), B_i)$. This proves the first claim. The second claim follows similar arguments. Since $P_i(x)$ is skew-symmetric for all x, we have that $\frac{\partial P_i(x)}{\partial x_j}$ is also skew-symmetric for all x. Then it is straightforward to check that differentiating the identity $S(x) = U^T(x)A(x)U(x)$ twice at a yields

$$\frac{\partial^2 S(a)}{\partial x_i \partial x_j} = P_{ij}^T(a) S(a) + S(a) P_{ij} + B_{ij}$$

where $P_{ij}(a) = \frac{\partial P_i(a)}{\partial x_j}$. Again, in view of Lemma 5.1, $P_{ij}(a)$ is unique and equal to $P_{ij}(a) = \mathcal{A}(S(a), B_{ij})$. This completes the proof. □

We now turn to the uniqueness question about the real Schur form $S(x)$ of $A(x)$ given in Theorem 5.1 (iii). ¿From the proof of Theorem 5.2, it is seen that the first and second order derivatives of $S(x)$ at a are uniquely defined by $S(a)$, $U(a)$ and the derivative at $A(x)$ at a. In fact, this result holds for higher order derivatives of $S(x)$ as well. Therefore, given two choices of analytic functions $\{U(x), S(x)\}$ and $\{\hat{U}(x), \hat{S}(x)\}$ in Theorem 5.1 (ii), it holds that $S(a) = \hat{S}(a)$ and all derivatives of $S(x)$ and $\hat{S}(x)$ at a coincide with each other. Thus, we must have $S(x) = \hat{S}(x)$ for all x. By the same argument, we can conclude also that $U(x) = \hat{U}(x)$ for all x. This proves the third claim of Theorem 5.1.

C.5.3 Derivatives of eigenvalues

Using the results on the derivatives of real Schur form, we are now ready to give the derivatives of the eigenvalues.

Theorem 5.3. Suppose that $A(a)$ has all distinct eigenvalues. Let $U(a)S(a)U^T(a)$ be a real Schur decomposition of $A(a)$ and assume $S(a) \in \mathcal{S}_n$. Also, let $U(x)$ and $S(x)$ be the unique analytic function given in Theorem 5.1 (ii) and (iii). Then, the first and second order derivatives of the eigenvalues of $A(x)$ at a can be computed as follows. If $\lambda(a)$ is a real eigenvalue of $A(a)$, i.e., $\lambda(a)$ is a diagonal element of $S(a)$, say, $s(a)$, then

$$\frac{\partial \lambda(a)}{\partial x_i} = \frac{\partial s(a)}{\partial x_j} \quad \text{and} \quad \frac{\partial^2 \lambda(a)}{\partial x_i \partial x_j} = \frac{\partial^2 s(a)}{\partial x_i \partial x_j} \qquad \text{for } i,j = 1,\ldots,m$$

Now, suppose that $\lambda(a)$ is a nonreal eigenvalue of $A(a)$ with positive imaginary part, i.e., $\lambda(a)$ is an eigenvalue of a 2×2 diagonal block of $S(a)$, say,

$$\begin{bmatrix} s_1(a) & s_2(a) \\ s_3(a) & s_4(a) \end{bmatrix} \tag{5.10}$$

Define

$$\gamma_i^{(k)} = \frac{\partial s_k(a)}{\partial x_i}, \qquad \gamma_{ij}^{(k)} = \frac{\partial^2 s_k(a)}{\partial x_i \partial x_j}, \qquad \tau = \sqrt{\frac{1}{\lambda(a) - \overline{\lambda}(a)}}$$

and $y_i = (s_1(a) - s_4(a))\left(\gamma_i^{(1)} - \gamma_i^{(4)}\right) + 2\, s_3(a)\, \gamma_i^{(2)}$. Then, for $i,j = 1,\ldots,m$, we have

$$\begin{aligned}
\frac{\partial \lambda(a)}{\partial x_i} &= \frac{1}{2}\left(\gamma_i^{(1)} + \gamma_i^{(4)} + y_i\, \tau\right) \\
\frac{\partial^2 \lambda(a)}{\partial x_i \partial x_j} &= \frac{1}{2}\left(\gamma_{ij}^{(1)} + \gamma_{ij}^{(4)} - y_i\, y_j\, \tau^3 + 2\, s_3(a)\, \gamma_{ij}^{(2)}\, \tau + \left(\gamma_i^{(1)} - \gamma_i^{(4)}\right)\left(\gamma_j^{(1)} - \gamma_j^{(4)}\right)\tau\right) \\
&\quad + \tau\, (s_1(a) - s_4(a))\left(\gamma_{ij}^{(1)} - \gamma_{ij}^{(4)}\right)
\end{aligned}$$

Proof. The result is obvious when $\lambda(a)$ is real. When $\lambda(a)$ is nonreal, one can first find its analytic expression in terms of the elements of the matrix in (5.10), and then apply the chain rule for its derivatives. □

C.5.4 Derivatives of $f_1(x)$ and $f_2(x)$

In this section, we give the expressions for various quantities that needed in Algorithm 4.1. They can be easily obtained by using the preceding results and the chain rule. Here, given a vector t, we denote by $(t)_i$ its i-th element, and given a matrix T, we denote by $(T)_{ij}$ its ij-th element. Also, to simplify the notation, we denote $\sum_{k=1}^{q-1} \sum_{l=k+1}^{q}$ by $\sum_{k,l}^{q}$.

$$
\begin{aligned}
g_1(x) &= \frac{1}{q}\sum_{k=1}^{q} t_k \\
H_1(x) &= \frac{1}{q}\sum_{k=1}^{q} T_k \\
g_2(x) &= \sum_{k,l}^{q} (\Re\lambda_k(x) - \Re\lambda_l(x))(t_k - t_l) \\
H_2(x) &= H_{21}(x) + H_{22}(x) \\
D(x,h) &= \sum_{k,l}^{q} \Big((T_k - T_l)h(t_k - t_l)^T + (t_k - t_l)h^T(T_k - T_l)\Big) \\
\hat{D}(x,\phi) &= \sum_{k,l}^{q} \Big(\Big((t_k - t_l)^T\phi I + (t_k - t_l)\phi^T\Big)(T_k - T_l)\Big) \\
H_{21}(x) &= \sum_{k,l}^{q} (t_k - t_l)(t_k - t_l)^T \\
H_{22}(x) &= \sum_{k,l}^{q} (\Re\lambda_k(x) - \Re\lambda_l(x))(T_k - T_l) \\
(t_k)_i &= \Re\frac{\partial\lambda_k(x)}{\partial x_i} \\
(T_k)_{ij} &= \Re\frac{\partial^2\lambda_k(x)}{\partial x_i \partial x_j}
\end{aligned}
$$

C.6 A numerical example

The proposed algorithm has been implemented in Matlab [6]. In this section, we present a simple example to illustrate its convergence property. The example is to minimize over $\mathbf{R}^2$ the largest real part of the eigenvalues of $A(x)$, where $A(x)$ is defined by

$$A(x) = A_0 + x_1A_1 + x_2A_2 + x_1x_2A_3 + x_1^2A_4 + x_2^2A_5$$

and the matrices A_i, $i = 0, \ldots, 5$ are given as follows.

$$
A_0 = \begin{bmatrix}
0.2 & 1.0 & -0.2 & 0.0 & -6.6 \\
0.7 & -3.1 & -1.5 & 5.3 & -1.5 \\
-1.4 & 2.8 & 4.6 & 0.9 & -6.3 \\
3.6 & 5.8 & -3.9 & 3.0 & -8.0 \\
0.3 & 9.1 & -3.2 & 2.0 & 1.7
\end{bmatrix}
\qquad
A_1 = \begin{bmatrix}
4.1 & 7.2 & 5.7 & -3.5 & -6.4 \\
-2.3 & 7.2 & 5.1 & -1.0 & -4.2 \\
0.8 & 1.9 & -6.4 & -4.6 & -0.3 \\
2.8 & 7.4 & -0.7 & 0.6 & -3.2 \\
-8.4 & 7.1 & -1.3 & -1.9 & 1.4
\end{bmatrix}
$$

$$A_2 = \begin{bmatrix} 4.4 & 2.3 & -1.1 & 6.6 & 7.8 \\ -4.6 & 9.2 & -6.7 & -1.8 & 5.9 \\ -3.1 & 0.2 & 11. & -8.0 & -7.0 \\ -3.6 & -2.8 & -4.2 & 13. & 2.7 \\ 3.1 & -2.6 & 4.3 & 0.9 & 10. \end{bmatrix} \quad A_3 = \begin{bmatrix} -9.4 & -3.5 & 1.5 & -1.5 & -4.8 \\ -1.6 & -0.9 & 0.4 & 0.8 & 0.8 \\ -2.1 & 0.5 & -3.8 & -3.6 & 5.8 \\ 7.7 & 2.3 & 3.7 & -6.6 & -2.7 \\ -4.9 & 0.3 & 2.5 & 2.0 & -1.5 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} -3.8 & 6.2 & -0.3 & -5.8 & -7.1 \\ -7.5 & 0.2 & 3.9 & -4.7 & 3.1 \\ -0.4 & -0.7 & -3.7 & -0.1 & -1.8 \\ -2.4 & -4.2 & 2.4 & 6.9 & -0.3 \\ -0.2 & -5.5 & 0.3 & 5.7 & 8.4 \end{bmatrix} \quad A_5 = \begin{bmatrix} 15. & -0.0 & -4.2 & -3.8 & 0.6 \\ 5.8 & 12. & 2.8 & -3.6 & 4.5 \\ 5.3 & -1.1 & 9.4 & 5.0 & 4.6 \\ -9.7 & 4.4 & 4.2 & 13. & 0.1 \\ 4.8 & -3.7 & -8.9 & 6.7 & 15. \end{bmatrix}$$

A local solution is

$$x^* = \begin{bmatrix} 0.14867145915551 \\ -0.38655872292658 \end{bmatrix}$$

which is obtained by trial and error. The corresponding eigenvalues of $A(x)$ at x^* are

$$\begin{bmatrix} \lambda_1(x^*) \\ \lambda_2(x^*) \\ \lambda_3(x^*) \\ \lambda_4(x^*) \\ \lambda_5(x^*) \end{bmatrix} = \begin{bmatrix} 3.96924962356182 + i\ 7.73645446194478 \\ 3.96924962356182 - i\ 7.73645446194478 \\ 3.96924962356181 \\ -1.78038425406443 \\ -10.02592118139874 \end{bmatrix}$$

Thus, the multiplicity of $\phi(x^*)$ is 3. Some level curves of $\phi(x)$ around x^* are given in Figure 6.1 (x^* is located at the center of the figure). Also, in Figure 6.2, solid lines are the level curves of $f_1(x)$ and the dotted line is the feasible set $\{x : f_2(x) = 0\}$.

We set $q = 3$ and choose the starting point $x = 0$ in running Algorithm 4.1. In order to ensure convergence in a larger neighborhood around x^*, we perform line search along $h(x^k)$ at each iteration. We choose l_k to be the smallest integer in $\{0, 1, 2, \cdots\}$ such that x^{k+1} defined by

$$x^{k+1} = x^k + (0.5)^{l_k}\, h(x^k)$$

strictly decreases the objective function, i.e., $\phi(x^{k+1}) < \phi(x^k)$. The test result is summarized in Table 6.1. In Table 6.1, the second column is the largest real part of the eigenvalues of $A(x^k)$; the third and fourth columns together is the optimality condition for a local minimizer (both values are zero at the minimizer); and the last column is the distance between x^k and the local minimizer x^*. The integer l_k used line search is one for the first iteration and zero for the rest. The result shows that the rate of convergence is indeed quadratic.

Acknowledgements. The work of this research was supported in part by NASA-Ames.

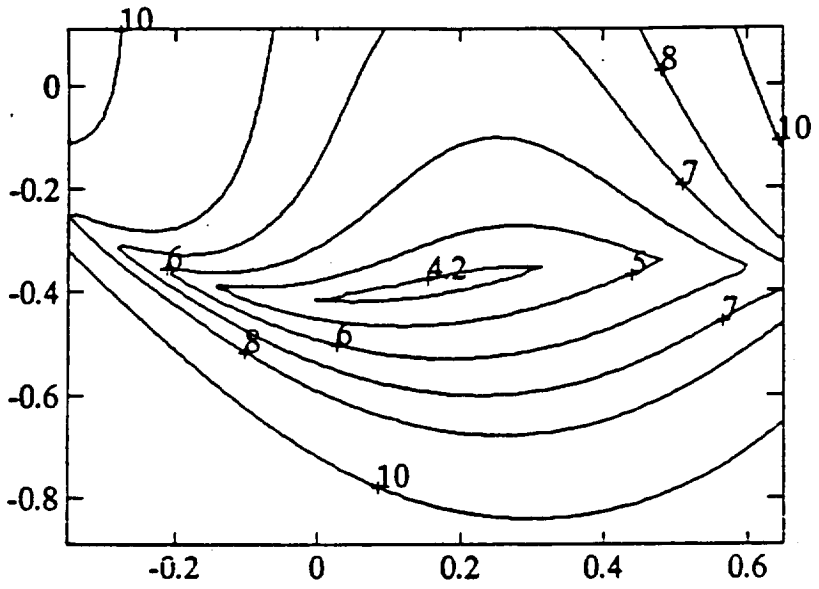

Figure 6.1. Level curves of $\phi(x)$.

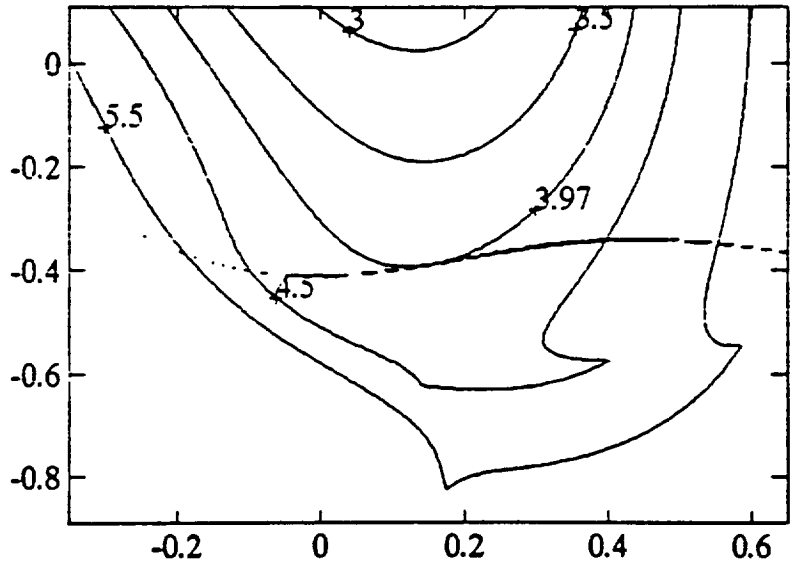

Figure 6.2. Solid lines are level crves of $f_1(x)$ and the dotted line is the set $\{x : f_2(x) = 0\}$.

Iter k	$\phi(x^k)$	$f_2(x^k)$	$\|N^T(x^k)g_1(x^k)\|$	$\|x^k - x^*\|$
0	6.960	2.6e+01	2.4e+01	4.1e−01
1	5.945	5.9e+01	9.9e−01	2.6e−01
2	5.754	7.9e+01	4.8e+00	1.3e−01
3	4.515	2.1e+00	5.2e+00	1.2e−01
4	4.000	2.2e−02	3.8e−02	3.0e−03
5	3.96958	2.5e−06	1.1e−03	2.7e−05
6	3.969249645	1.1e−14	1.4e−08	2.3e−09
7	3.96924962356182	7.9e−31	1.2e−14	1.0e−15

Table 6.1

References

[1] F.H. Clarke, *Optimization and Nonsmooth Analysis*, J. Wiley & Sons, 1983.

[2] M.K.H. Fan, "A quadratically convergent local algorithm on minimizing the largest eigenvalue of a symmetric matrix," to appear in the special issue of *Linear Algebra and Its Applications on Numerical Linear Algebra Methods in Control, Signals and Systems*, 1993.

[3] R.A. Horn and C.R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[4] T. Kato, *Perturbation Theory for Linear Operators*, 2nd edition, Berlin Springer, 1976.

[5] P. Lancaster and M. Tismenetsky, *The Theory of Matrices*, 2nd ed., Academic Press, 1985.

[6] C. Moler, J. Little and S. Bangert, *PRO-MATLAB User's Guide*, The MathWorks, Inc., Sherborn, MA, 1987.

[7] E. Polak and Y. Wardi, "Nondifferentiable optimization algorithm for the design of control systems subject to singular value inequalities," *Automatica*, vol. 18, no. 3, pp. 267-283, 1982.

[8] A. Shapiro and M.K.H. Fan, "On eigenvalue optimization", submitted to SIAM J. on Optimization, March 1993.

[9] N.-K. Tsing, M.K.H. Fan and E.I. Verriest, "On analyticity of functions involving eigenvalues," to appear in *Linear Algebra and Its Applications*, 1993.